

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Informática

PROYECTO FIN DE CARRERA

Generador Automático de un Sistema de Publicación Web para Institutos de Enseñanza Secundaria

Autor: Juan Carlos Parrilla Peláez

Director: Carlos González Morcillo

Septiembre, 2004

TRIBUNAL:

Presidente:

Vocal:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL	SECRETARIO
Fdo.:	Fdo.:	Fdo.:

A mi mujer, María Dolores, y a mis hijos, Carlos y Pablo. Ellos son los verdaderos sufridores de mis inquietudes y de mi afán de superación.

Agradecimientos

A Carlos González, por su apoyo, confianza e ideas para el diseño del interfaz de usuario.

Al grupo de profesores de los I.E.S. Virgen de Gracia de Puertollano y Aljada de Murcia que han colaborado en el estudio de requisitos. Sus ideas, pruebas y críticas constructivas han sido determinantes en el resultado final.

Resumen

Estar presente en la red Internet es actualmente, más que una opción, una obligación. Este proyecto pretende unificar los sitios Web correspondientes a los Institutos de Enseñanza Secundaria desarrollando un generador automático que permita a los Centros que lo deseen disponer de un sistema de publicación Web personalizado que satisfaga sus necesidades reales sin necesidad de tener conocimientos que no vayan más allá que los que posee cualquier internauta. El generador se implementa con Zope, un servidor de aplicaciones orientado a objetos de software libre.

Índice

Lista de Figuras

1.- INTRODUCCIÓN.

Internet se ha convertido en el sistema más importante de publicación-obtención de información de la sociedad actual. Hace unos años, las redes de comunicación de área extensa se utilizaban principalmente para servicios de transferencia de archivos, correo electrónico, conexión remota a ordenadores centrales, etc. Uno de los aspectos que motivaban esta limitación era que los usuarios que tenían acceso a estos servicios pertenecían a sectores puntuales de la sociedad.

Actualmente, las posibilidades han evolucionado a un nivel en el que es difícil imaginar aplicaciones de la vida cotidiana que no tengan cabida en la red. Este avance ha sido motivado, en gran medida, por el aumento progresivo de usuarios: Internet es accesible desde todos los organismos públicos y privados, todas las grandes y medianas compañías, gran parte de las pequeñas empresas y, en general, en una gran parte de los hogares de los países desarrollados.

¿O quizás el número de usuarios de Internet se ha incrementado como consecuencia de los avances de la tecnologías de desarrollo de sistemas de publicación de información?

Sea cual fuere el motivo y cual la consecuencia, la realidad es que Internet es una herramienta de trabajo imprescindible en las empresas modernas y en los organismos públicos y privados.

Para facilitar la presencia de los Centros de Enseñanza en Internet, las Consejerías de Educación las Comunidades Autónomas conceden a cada centro un espacio para la publicación de su página Web, le asigna una URL (por ejemplo, http://www.educarm.es/ies-aljada) y facilita una dirección de correo electrónico (por ejemplo, ies-aljada@educarm.es). A partir de aquí, cada centro se las ingenia para desarrollar y publicar su Web (contratando los servicios de alguna empresa privada, aprovechando la buena voluntad e interés en el tema de miembros de la Comunidad Educativa, convocando un concurso, o en el peor de los casos, ignorando la posibilidad de usar y explotar las posibilidades de la presencia en la red).

Con estos precedentes, es posible encontrar sitios Web completamente dispares. Difieren en el aspecto (colores, fuentes, marcos, etc.), en el contenido (desde una página única con los datos postales del centro hasta contenidos muy completos), en el sistema de navegación (mediante enlaces de texto en forma arbórea, enlaces de texto sin orden, enlaces mediante imágenes, etc.), en los servicios dinámicos (mientras que en la mayoría no existen, algunos permiten consultar expedientes, otros permiten reservar plaza, etc), en las herramientas de desarrollo de los sitios (HTML, ASP, JavaScript, herramientas WYSIWYG etc.), en los usuarios potenciales y en general, casi en todo. Es difícil encontrar dos sitios Web de dos Institutos de Enseñanza Secundaria que se parezcan en algo.

Existen aspectos comunes a la mayoría de las páginas Web de los centros: llevan asociado un trabajo perpetuo y no reconocido de alguna(s) persona(s) de la Comunidad Educativa (para añadir información al contenido del sitio hay que "molestar" al responsable-encargado-administrador-operador del Web), tienen limitada la adecuada mantenibilidad del sitio, gestionan "manualmente" la información, etc.

1.1.- ESTRUCTURA DEL DOCUMENTO.

Este proyecto fin de carrera se ha estructurado en siete capítulos y dos anexos.

El capítulo 1 es una introducción al tema tratado en la que se intenta mostrar la problemática existente en la publicación Web en los Institutos de Enseñanza Secundaria como justificación al trabajo abordado en el resto del proyecto. También incluye la estructura del resto del documento.

El capítulo 2 enumeran los objetivos que pretenden conseguirse con el proyecto. Existe un objetivo principal y muy concreto que es la implementación del generador automático. Para conseguirlo, existen otros objetivos necesarios que se plantean simultáneamente.

El capítulo 3 realiza un repaso sobre la situación del estado del arte en las disciplinas asociadas a los objetivos del proyecto. Teniendo en cuenta que el objetivo principal es el desarrollo de un producto software, es imprescindible realizar un estudio teórico de la Ingeniería del Software. El hecho de que la aplicación desarrollada sea cliente – servidor basada en Web implica el estudio de las metodologías, arquitecturas, principios y tecnologías existentes. Finalmente, se muestran las características de los principales sistemas gestores de contenidos del mercado.

El capítulo 4 presenta cómo se ha solucionado el problema, realizando un resumen del proceso de Ingeniería de Software asociada al desarrollo de la aplicación.

El capítulo 5 es una presentación del resultado obtenido con el proyecto: GASPWIES. Se describe cómo realizar la instalación de la aplicación, se ejemplifica con un sitio Web en funcionamiento y se indica cómo utilizar el manual de usuario.

El capítulo 6 expone las conclusiones obtenidas con el desarrollo del proyecto y se plantean propuestas de mejora y ampliación de la aplicación.

El capítulo 7 incluye un listado alfabético de la bibliografía y referencias Web que se han consultado para la realización de este proyecto fin de carrera.

El anexo I contiene una relación con el código fuente de algunas plantillas y *scripts* que se consideran más interesantes didácticamente.

El anexo II muestra las imágenes de la principal información que ofrece un sitio Web en funcionamiento.

2.- OBJETIVOS DEL PROYECTO.

El principal objetivo de este proyecto fin de carrera es *prototipar* el sitio Web de un Instituto de Enseñanza Secundaria de forma que cualquier Centro pueda disponer de un sistema de publicación en Internet para proporcionar el mayor número posible de servicios a los miembros de la Comunidad Educativa, incluso para servir a usuarios que sin pertenecer a dicha Comunidad, puedan estar interesados en determinados aspectos del Centro (futuros alumnos, padres de futuros alumnos, miembros de otras Comunidades Educativas, etc.).

El enorme conjunto de documentos que constituyen los sitios Web actuales se sustituirá por un sistema que permitirá administrar un portal con publicación en línea, que en definitiva, es la necesidad que se plantea en los Centros.

Para conseguirlo, se desarrollará un producto software que abarque genéricamente las actividades cotidianas de los Centros. El producto da nombre al proyecto: *Generador Automático de un Sistema de Publicación Web para Institutos de Enseñanza Secundaria (GASPWIES)*.

GASPWIES será un sistema gestor de contenidos que permitirá a los usuarios de cada portal realizar publicaciones de noticias de interés clasificadas en secciones para los miembros de la Comunidad Educativa. Facilitará la puesta en marcha de páginas Web de los Departamentos Didácticos en los que se organiza cada Centro. Permitirá a los profesores disponer de su página Web personal, en la que podrán publicar contenido para los alumnos y gestionar sus fichas personales.

Todos los sitios Web generados con GASPWIES tendrán la misma estructura y funcionalidad. Sin embargo, cada Centro podrá personalizar el aspecto del portal, el contenido específico del Centro como es su historia, sus instalaciones, su claustro de profesores, su equipo directivo, etc.

Para la consecución de este objetivo principal, se han planteado otros objetivos inherentes a desarrollo de una aplicación software basada en Web:

- Realizar un repaso al estado del arte en la Ingeniería del Software, profundizando en las principales metodologías de desarrollo para aplicaciones basadas en la Web.
- Exponer la tendencia actual en el diseño arquitectural de las aplicaciones Web y en las tecnologías de desarrollo.
- Hacer una reflexión sobre los principios básicos que garantizan el éxito de un sitio Web.
- Analizar las características un sistema gestor de contenidos, comparando algunas de las alternativas existentes en el mercado.

3.- ANTECEDENTES, ESTADO DE LA CUESTIÓN.

3.1.- INGENIERÍA DEL SOFTWARE.

La Ingeniería del Software es "*una disciplina o área de la Informática que ofrece métodos y técnicas para desarrollar y mantener software de calidad que resuelven problemas de todo tipo*" [PRESSMAN 98].

La Ingeniería del Software trata con áreas muy diversas de la Informática, tales como construcción de compiladores, sistemas operativos o desarrollos en Intranet/Internet, abordando todas las fases del ciclo de vida del desarrollo de cualquier tipo de sistemas de información y es aplicable a una infinidad de áreas tales como: negocios, investigación científica, medicina, producción, logística, banca, control de tráfico, meteorología, el mundo del derecho, la red de redes Internet, redes Intranet, etc.

Muchos autores han definido el término Ingeniería del Software. A continuación se muestran dos de las más extendidas y reconocidas:

- *Es el estudio de los principios y metodologías para desarrollo y mantenimiento de sistemas de software*. Zelkowitz, M.
- *Es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento del software, es decir, la aplicación de la ingeniería al software.* IEEE, 1993.

Este apartado pretende realizar una introducción a los paradigmas en Ingeniería del Software, desde sus inicios hasta el actual Proceso Unificado de Desarrollo de Software o también llamado Proceso Unificado de Modelado.

3.1.1.- Modelos de proceso de software.

Para resolver un problema, los ingenieros del software deben incorporar una estrategia de desarrollo que acompañe al proceso, métodos y fases genéricas (análisis, diseño, implementación y pruebas). Esta estrategia se denomina paradigma de Ingeniería

del Software o modelo de proceso. La selección del paradigma se realiza en función de la naturaleza del proyecto y otros factores como los controles y entregas requeridas. Los principales modelos de proceso de software son los siguientes:

a) *Modelo lineal secuencial*: También llamado ciclo de vida básico o modelo en cascada sugiere un enfoque sistemático y secuencial del desarrollo del software que comienza en un nivel de sistemas y progresa con el análisis, diseño, codificación, pruebas y mantenimiento.

Este modelo es el más antiguo y el más extensamente utilizado en la Ingeniería del Software. Sin embargo, la crítica ha puesto en duda su eficacia. Algunos problemas que se le achacan son:

- No refleja realmente el proceso de desarrollo del software.
- Se tarda mucho tiempo en pasar por todo el ciclo.
- Perpetúa el fracaso de la industria del software en su comunicación con el usuario final.
- El mantenimiento se realiza en el código fuente.
- Las revisiones de proyectos de gran complejidad son muy difíciles.
- Impone una estructura de gestión de proyectos.

b) *Modelo de construcción de prototipos*: Este modelo comienza con la recolección de requisitos y se realiza un diseño "rápido". Este diseño se centra en una representación de esos aspectos del software que serán visibles para el cliente. Esto lleva a la construcción de un prototipo que evalúa el cliente y se refina según la opinión del cliente.

Este modelo también tiene problemas:

- El cliente ve funcionando lo que para él es la primera versión del prototipo que ha sido construido con "plastilina y alambres", y puede desilusionarse al decirle que el sistema aún no ha sido construido.
- El desarrollador puede caer en la tentación de ampliar el prototipo para construir el sistema final sin tener en cuenta los compromisos de calidad y de mantenimiento que tiene con el cliente.

c) *Modelo incremental*: Combina elementos del modelo lineal secuencial con la filosofía interactiva de construcción de prototipos. Este modelo aplica secuencias lineales de la misma forma que progresa el tiempo en el calendario. Cada secuencia lineal produce un incremento del software.

Los principales problemas de este modelo son:

- Difícil de evaluar el coste total.
- Difícil de aplicar a sistemas transaccionales que tienden a ser integrados y a operar como un todo.
- Requiere gestores experimentados.
- Los errores en los requisitos se detectan tarde.

d) *Modelo en espiral*: es un modelo evolutivo que acompaña la naturaleza interactiva de la construcción de prototipos con los aspectos controlados y sistemáticos del modelo lineal secuencial. El software se desarrolla en una serie de versiones incrementales. Se divide en *actividades estructurales*.

3.1.2.- Metodologías de desarrollo de software orientadas a objetos.

Los objetos forman el mundo real. Existen en todas las facetas y disciplinas de la vida. No es extraño que surja una propuesta de desarrollo de software pensando en los objetos que forman parte del sistema que se pretende modelar. A pesar de que el primer enfoque orientado a objetos para desarrollo de software se propuso a finales de los años 60, las tecnologías orientadas a objetos empezaron a considerarse y generalizarse en su uso a mediados de los años 90. Actualmente, la Ingeniería del Software orientada a objetos ha sustituido prácticamente en su totalidad a la visión estructurada de desarrollo de software.

Los motivos de esta implantación pueden ser varios, pero sin duda, su característica de facilitar la reutilización de componentes software y el consecuente desarrollo de software más rápido y de calidad es uno de los principales motivos. La reutilización implica otros beneficios como la mejora en el mantenimiento del software, la reducción de efectos laterales, la facilidad para adaptar y escalar sistemas, etc.

A continuación se presentan los fundamentos de tres metodologías orientadas objetos: OMT, Booch y OOSE. Sus autores, Rumbaugh, Booch y Jacobson, respectivamente, han unificado las ideas más importantes de las tres metodologías para crear el *proceso unificado de desarrollo de software.*

3.1.2.1.- *OMT (Object Model Tecnology).*

La metodología OMT es una técnica de modelado de objetos, desarrollada por James Rumbaugh, que es uno de los precursores del Lenguaje Unificado de Modelado (UML). Está definida como una de las metodologías de la Ingeniería del Software aplicable al desarrollo orientado a objetos en las fases de análisis y diseño. La fase de análisis comienza con la declaración del problema e incluye una lista de objetivos (metas) y conceptos claves definitivos definidos para el dominio del problema a resolver. La declaración del problema se "expande" después en tres modelos:

- Modelo de objetos: representa los objetos del sistema.
- Modelo dinámico: representa la interacción entre esos objetos representados como eventos, estados y transiciones.
- Modelo funcional: representa los métodos del sistema desde la perspectiva de flujo de datos.

La metodología OMT lleva asociada parte del modelo secuencial lineal en el sentido de que la primera fase es la de análisis, seguida por el diseño. En cada fase, se hacen aproximaciones iterativas entre los pequeños pasos a seguir. Su principal criterio es hacer enfásis en las fases de análisis y diseño para lograr una primera entrega del producto. OMT no hace prioritarias las fases de implementación, pruebas u otras del ciclo de vida [PRESSMAN 98].

Esta metodología es aplicable en varios aspectos de implementación incluyendo archivos, bases de datos relacionales y orientadas a objetos. Se construye alrededor de descripciones de estructuras de datos, constantes y sistemas de procesos de transacciones. Hace enfásis en especificaciones declarativas de la información, captura de manera

transparente los requisitos, especificaciones imperativas para poder descender prematuramente en el diseño y declaraciones que permiten optimizar los estados.

Las fases que se proponen en OMT son las siguientes:

a) *Conceptualización*: Es una fase inicial (algunos autores la incluyen en la siguiente) en la cual el desarrollo comienza con el análisis de la empresa o negocio, la visión que tienen los usuarios del sistema y la formulación de requisitos. La conceptualización se hace frecuentemente por la re-ingeniería de procesos del negocio, es decir, se pretende tener una observación crítica de los procesos de la empresa y su impacto económico.

b) *Análisis*: genera diagramas del modelo de objetos, diagramas de estado, diagramas de eventos de flujo y diagramas de flujos de datos. Los requisitos establecidos durante la conceptualización son revisados y analizados para la construcción del modelo real. La meta del análisis es especificar las necesidades que debe satisfacer el sistema desarrollado. Pueden existir diversas fuentes de información que pueden ser útiles en esta etapa.

   En esta fase se determina el modelo del objeto, se hace una tentativa de clases, se eliminan las clases irrelevantes, se definen las posibles asociaciones entre las clases y se hace la refinación de clases eliminando las redundantes. Posteriormente se hace una tentativa de atributos de objetos y enlaces.

   Una vez determinados los objetos del sistema, se hace la depuración del modelo y se busca un nivel de abstracción para modelar subsistemas, para buscar un sistema tangible y sólido. Tras tener definido el modelo, se introduce la noción de transacción, que es una forma de modelar procesos o describir cambios y flujos de datos; una vez definido el flujo de datos se define el diccionario de datos de todas las entidades modeladas.

c) *Diseño*: se define la arquitectura completa del sistema. Primero el sistema se organiza en subsistemas que están asignados a ciertos procesos y tareas, tomando en cuenta la colaboración y concurrencia entre ellos. Luego, se establece el almacenamiento persistente de datos. Después, se examinan las situaciones límite para ayudar a establecer las prioridades.

Posteriormente, y tras la fase de diseño del sistema se realiza la fase de diseño de objetos donde se establece el plan de implementación. Se definen las clases de objetos y sus algoritmos, poniendo especial atención con la optimización y persistencia de datos. Se definen cuestiones de herencia, asociaciones, agregaciones y valores por omisión.

Algunos autores dividen esta fase en dos: diseño del sistema y diseño de objetos.

d) *Implementación*. Las clases de objetos y relaciones desarrolladas durante el análisis de objetos se traducen finalmente a una implementación concreta. Durante la fase de implementación es importante tener en cuenta los principios de la Ingeniería del Software de forma que la correspondencia con el diseño sea directa y el sistema implementado sea flexible y extensible. No tiene sentido utilizar análisis y diseño orientado a objetos para potenciar la reutilización de código y la correspondencia entre el dominio del problema y el sistema informático, si luego se pierden estas ventajas con una implementación de mala calidad.

Existen en el mercado herramientas que permiten el uso de la metodología OMT. SmartDraw está diseñada especialmente para su uso. Otras herramientas que soportan metodología OMT son: Rational Rose, System Architect, Poseidón, Power Builder, ObjectMarker, BOCS, ObjectTeam, OMTool, etc.

3.1.2.2.- *Booch.*

Su autor es Grady Booch. La metodología Booch se enfoca principalmente al diseño de estado de un proyecto [BMODEL]. Booch describe una serie de propiedades generales de los sistemas complejos bien estructurados. Los sistemas construidos con una metodología de análisis y diseño orientado a objetos deben satisfacer estas propiedades.

En el análisis y diseño orientado a objetos, el dominio del problema se modela a partir de dos perspectivas distintas. La estructura lógica del sistema y la estructura física. Para cada perspectiva (dinámica y estática) se modela la semántica. La metodología Booch

define diferentes modelos para la descripción de un sistema. El modelo lógico (dominio del problema) se representa en la estructura clase-objeto. En el diagrama de clase, se construye la arquitectura y el modelo estático. El diagrama de objeto, representa la interacción de clases entre sí, captura algunos momentos de la vida del sistema y ayuda en la descripción del modelo dinámico.

En general, no difiere demasiado de la metodología OMT. También es iterativa e incremental. Una diferencia importante es su distinción entre:

- Procesos *macro*: se establecen los requisitos del núcleo (conceptualización), se desarrolla un modelo del comportamiento deseado (análisis), se crea la arquitectura (diseño), se realiza la implementación y se prueba.
- Procesos *micro*: se identifican las clases y objetos a cierto nivel de abstracción, se identifica la semántica de estas clases y objetos, se identifican las relaciones entre ellas y se especifica el interfaz. Después se realiza la implementación de las clases y objetos.

3.1.2.3.- <u>*OOSE (Object Oriented Software Engineering).*</u>

Su autor es Ivar Jacobson. Plantea una metodología de desarrollo de aplicaciones orientada a objetos. Propone el desarrollo de sistemas basado en el uso de distintos modelos. Proporciona un soporte para el diseño creativo de productos de software. Las actividades consisten en la transformación de un conjunto de requisitos en un plan estructurado de construcción y un plan de acción.

El ciclo de vida se basa en la sucesión de modelos:

a) *Modelo de requisitos*: Se limita el alcance del sistema y define la funcionalidad que este debe ofrecer. Este modelo debería servir como contrato entre el usuario final del software y el equipo de desarrollo. Este enfoque plantea que el sistema debe construirse desde la perspectiva del usuario de forma que sea más fácil de entender y discutir. Consta de tres partes:
    - *Modelo de casos de uso*: especifica la funcionalidad que se debería implementar en el sistema. Este modelo usa actores para representar los

roles que los usuarios pueden tomar y casos de uso que representan lo que el usuario debería poder hacer con el sistema. Cada caso de uso es un camino de eventos en el sistema visto desde la perspectiva del usuario.

- *Descripción de interfaces*: Si es necesario, se pueden incluir descripciones sobre el interfaz del usuario que especifiquen en detalle como se verán cuando se vayan ejecutando los casos de uso.
- *Modelo de objetos del dominio del problema*: se utiliza este modelo compuesto de objetos para lograr una visión conceptual del problema y para lograr un mejor entendimiento del mismo. Los objetos representan ocurrencias en el dominio del problema.

b) *Modelo de análisis*: especifica el comportamiento funcional del sistema bajo prácticamente circunstancias ideales y sin aludir a un entorno particular de implementación. Tras desarrollar el modelo de requisitos y la aprobación del usuario, se puede iniciar el desarrollo del sistema. La información para este sistema se enfoca en la captura de:

- Información: especifica la información de ayuda en el sistema. Así como describe el estado interno del sistema.
- Comportamiento: especifica el comportamiento que adopta el sistema. Especifica cuando y como el sistema cambia de estado.
- Presentación: detalla la presentación del sistema al mundo exterior. Existen varios tipos de objetos usados para la estructura del sistema en el modelo de análisis.

Cada objeto al menos captura dos de las tres dimensiones del modelo de análisis, sin embargo cada uno de ellos tiene cierta inclinación hacia una de las dimensiones.

c) *Modelo de diseño de objetos*: es un refinamiento y formalización del modelo de análisis. Una diferencia entre el modelo de análisis y el modelo de diseño es que el modelo de análisis debe ser visto como un modelo conceptual o lógico del sistema, y el modelo de diseño contiene el código, por lo cual el modelo de

diseño deberá ser una representación de la manera como se escribe y estructura el código fuente. Para cada objeto en el modelo de análisis, se asigna un bloque en el modelo de diseño. Un bloque puede ser de interfaz, de entidad o de control. Cuando se tiene que crear la estructura del bloque, se dibuja un diagrama de interacción para mostrar como se comunican los bloques. Normalmente se dibuja un diagrama de interacción para cada caso de uso.

d) *Modelo de implementación*: el modelo de diseño se traduce al lenguaje de implementación.

e) *Modelo de prueba*: describe el estado de resultados de la prueba. El modelo de requisitos representa una herramienta potente de prueba, al probar cada caso de uso, se verifica que los objetos se comunica correctamente en dicho caso de uso. De manera similar se verifica el interfaz de usuario.

3.1.3.- El Proceso Unificado de Desarrollo de Software.

En la actualidad, se tiende a construir grandes sistemas que cada vez son más complejos, en gran parte, debido a que la potencia de los ordenadores crece desmesuradamente y los usuarios les exigen más. Esta tendencia se ha visto afectada por el uso creciente de Internet para el intercambio de información y el comercio electrónico.

En Ingeniería del Software el objetivo es construir un producto software o mejorar uno existente. En ambos casos, el protagonista principal es el proceso. Un proceso define *quién* está haciendo *qué*, *cuando* y *cómo* alcanzar un determinado objetivo.

Un proceso de desarrollo de software debe ser capaz de evolucionar durante años limitando su alcance, en cada momento del tiempo, a las realidades que permitan las tecnologías, herramientas, personas y patrones de organización.

El problema de la producción de software se reduce a la dificultad que afrontan los desarrolladores para coordinar las múltiples cadenas de trabajo de un gran proyecto. Estos

desarrolladores necesitan una forma coordinada de trabajar: necesitan un proceso que integre las múltiples facetas del desarrollo. Necesitan un método común, un ***proceso*** que proporcione una guía para ordenar las actividades de un equipo, dirija las tareas de cada desarrollador por separado y del equipo como un todo, especifique los artefactos que deben desarrollarse y ofrezca criterios para el control y la medición de los productos y actividades del proyecto.

El Proceso Unificado es un proceso de desarrollo de software: un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software. Además, es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, diferentes tipos de organización, diferentes niveles de aptitud y diferentes tamaños de proyecto. Finalmente, es el resultado de la integración de las metodologías orientadas a objetos tratadas anteriormente junto con la experiencia de sus autores.

Las características principales del Proceso Unificado son:

a) *Está dirigido por casos de uso*: Un sistema software nace para dar servicio a los usuarios (humanos y otros sistemas). Un usuario se considera alguien o algo que interactúa con el sistema que se desarrolla. En [JACOBSON 00] se ejemplifica esta interacción con la inserción de una tarjeta electrónica en un cajero automático. El propietario de la tarjeta responde a las preguntas que la máquina le hace en pantalla (pin, idioma, operación que desea, etc.), y obtiene un resultado (dinero en efectivo, un listado, una confirmación, etc.). En definitiva, como respuesta a la inserción de la tarjeta y las respuestas del usuario, el sistema realiza una *secuencia de acciones* que proporcionan al usuario un resultado.

   Una interacción de este tipo es un *caso de uso*. Un caso de uso es un fragmento de funcionalidad del sistema que proporciona al usuario un resultado importante. Los casos de uso representan los requisitos funcionales del sistema. El conjunto de todos los casos de uso representan el *modelo de casos de uso*, que describe la funcionalidad total del sistema. En el modelo estructurado de desarrollo de software, la especificación funcional contesta a la pregunta, ¿qué debe hacer el sistema?. La estrategia de los casos de uso permite contestar a la pregunta, ¿qué

debe hacer el sistema para cada usuario?. La adición de estas tres palabras permiten particularizar la funcionalidad del sistema para los distintos tipos de usuarios del sistema.

Los casos de uso no sólo son una herramienta para especificar los requisitos de un sistema; también guían su diseño, implementación y prueba: guían el proceso de desarrollo. *Dirigido por casos de uso* quiere decir que el proceso de desarrollo sigue un hilo que avanza a través de los flujos de trabajo que parten de los casos de uso.

b) *Está centrado en la arquitectura*: El papel de la arquitectura software es parecido al papel que juega la arquitectura en la construcción de edificios. La arquitectura en un sistema software se describe mediante diferentes vistas del sistema en construcción (estructura del edificio, fontanería, sistema de climatización, electricidad, etc.).

El concepto de arquitectura software incluye los aspectos estáticos y dinámicos más significativos del sistema y se refleja en los casos de uso. La arquitectura está influida por factores como la plataforma sobre la que se explotará el sistema (hardware, sistema operativo, sistema gestor de bases de datos, protocolos de red, etc.), los módulos de software disponibles para reutilizar, consideraciones de implantación, rendimiento, usabilidad, etc. La arquitectura es una vista del diseño completo con las características más importantes resaltadas, dejando los detalles a un lado.

En resumen, los arquitectos del sistema:

- Comienzan creando un borrador que contiene la parte que no es específica de los casos de uso, por ejemplo, la plataforma.
- Posteriormente trabajan con un conjunto de casos de uso (los elegidos para representar las funciones claves del sistema). Cada caso de uso elegido se especifica en detalle y se realiza en términos de subsistemas, clases y componentes.
- A la vez que se detallan casos de uso y éstos maduran, se descubre más de la arquitectura.
- El proceso continúa hasta considerar que la arquitectura es estable.

c) *Es iterativo e incremental:* Como principio básico de desarrollo de sistemas, es conveniente dividir el proyecto de un sistema en partes más pequeñas. Cada parte se considera una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en el flujo de trabajo y es conveniente que estén controladas, y los incrementos, al crecimiento del producto.

Los desarrolladores basan la selección de lo que se implementará en una iteración en dos factores: la iteración trata un grupo de casos de uso que juntos amplían la utilidad del producto desarrollado hasta el momento y trata los riesgos más importantes.

Algunos de los beneficios de un proceso iterativo controlado son:

- Reduce el coste del riesgo al coste de un solo incremento.
- Reduce el riesgo de no cumplir el calendario previsto para sacar producto al mercado.
- Acelera el ritmo del esfuerzo de desarrollo (los desarrolladores obtienen resultados claros a corto plazo).
- Reconoce una realidad antes ignorada: las necesidades del usuario y sus correspondientes requisitos no pueden definirse completamente al principio del desarrollo.

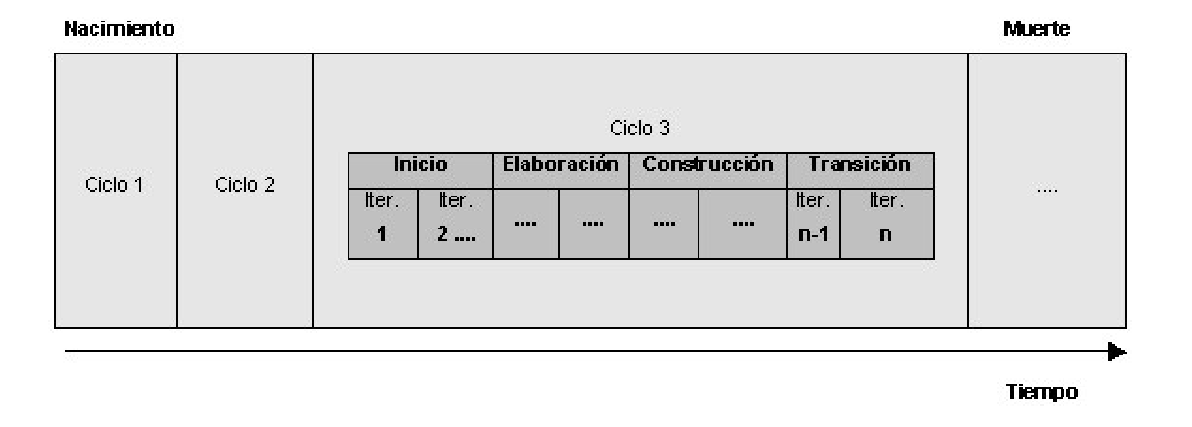


Figura 3.1. Vida del Proceso Unificado.

3.1.3.1.- La vida del Proceso Unificado.

La vida de un sistema se divide en una serie de ciclos en los que se repite el Proceso Unificado. Cada ciclo consta de cuatro fases: inicio, elaboración, construcción y transición. Cada fase se subdivide en iteraciones. Al final del ciclo se obtiene una versión del producto.

En la *fase de inicio* se desarrolla una descripción del producto final a partir de una buena idea y se presenta el análisis de negocio para el producto. Se describen las principales funciones del sistema para los usuarios más importantes, se hace un esquema-boceto de la arquitectura del sistema y se estudia el plan del proyecto y su coste.

En la *fase de elaboración* se especifican en detalle la mayoría de los casos de uso del producto y se diseña la arquitectura del sistema. Al final de la fase, el director del proyecto está en disposición de planificar las actividades y estimar los recursos necesarios para terminar el proyecto.

En la *fase de construcción* se crea el producto. Al final de la fase, el producto contiene todos los casos de uso especificados por el cliente. Sin embargo, puede que no esté libre de defectos.

La *fase de transición* cubre el período durante el cual el producto es una versión "beta". Un número reducido de usuarios con experiencia prueba el producto e informa de deficiencias y defectos.

3.1.3.2.- Los flujos de trabajo fundamentales.

En cada ciclo en la vida del sistema y durante las cuatro fases vistas con anterioridad tienen lugar cinco flujos de trabajo fundamentalmente con un mayor o menor peso según sea la fase y el flujo de trabajo.

A continuación se resumen las actividades a realizar en cada uno de los cinco flujos de trabajo:

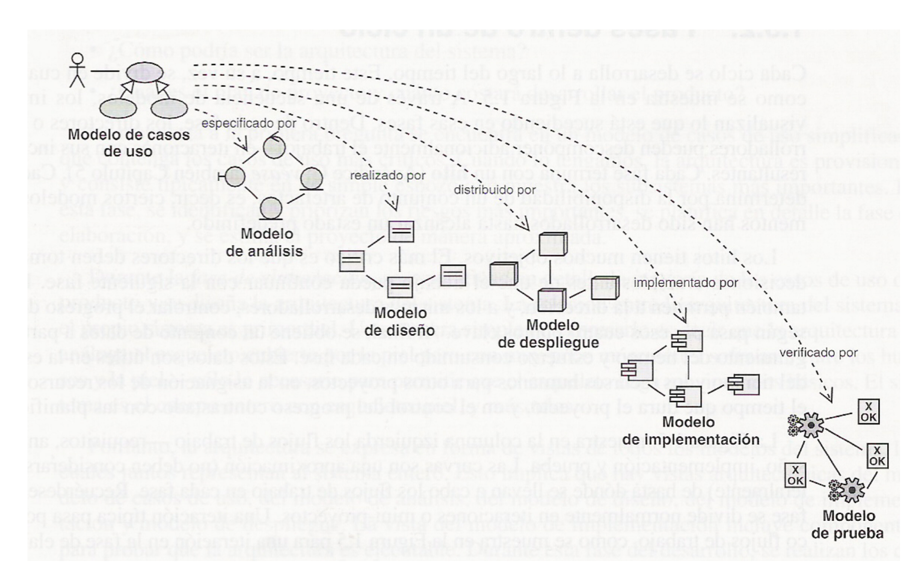


Figura 3.2. Modelo del Proceso Unificado.

a) *Captura de requisitos*: Es un proceso de descubrimiento que persigue averiguar qué se debe construir. Es seguramente el proceso más dificultoso debido a la importancia del papel que juegan los usuarios (clientes) ya que, habitualmente, éstos son una fuente no concreta de información (además, normalmente, no saben qué parte de su trabajo puede mecanizarse y cuál no) y debido a las implicaciones que tiene este papel de los usuarios (los flujos posteriores se basan en éste).

El objetivo fundamental de este flujo es guiar el desarrollo hacia el sistema correcto mediante una descripción de los requisitos del sistema suficientemente buena, de forma que pueda llegarse a un acuerdo con el cliente de lo que debe y lo que no debe hacer el sistema.

Este flujo de trabajo debe realizarse llevando a cabo una enumeración de los requisitos candidatos, la comprensión del contexto del sistema y la captura de los requisitos funcionales y no funcionales.

El artefacto[1] fundamental que se utiliza en la captura de requisitos es el modelo de casos de uso que contiene actores (suelen corresponderse con trabajadores-usuarios en una empresa o negocio; se suelen agrupar por roles), casos de uso y sus relaciones.

---

[1] Artefacto es un término general para cualquier tipo de información intermedia creada, producida, cambiada o utilizada por los trabajadores en el desarrollo del sistema. Ejemplos: diagramas UML, bocetos de interfaz de usuario, prototipos, componentes, etc. [JACOBSON 00]

b) *Análisis:* Se analizan los requisitos que se han descrito en el flujo anterior y que están reflejados en el modelo de casos de uso, refinándolos y estructurándolos. Para ello, se utiliza el lenguaje de los desarrolladores (en el flujo anterior, se utiliza el del cliente), lo que permite formalizar los requisitos, profundizar en aspectos internos del sistema y resolver aspectos relativos a la interferencia de los casos de uso.

   El análisis proporciona el modelo de análisis que sirve como entrada fundamental para dar forma al sistema en su totalidad.

   En resumen, el modelo de análisis es importante porque:

   - Ofrece una especificación más precisa de los requisitos que la obtenida en el modelo de casos de uso.
   - Se describe utilizando el lenguaje de los desarrolladores, y puede por tanto introducir un mayor formalismo y ser utilizado para razonar sobre los funcionamientos internos del sistema.
   - Estructura los requisitos de un modo que facilita su comprensión, su preparación, su modificación, y en general, su mantenimiento.
   - Puede considerarse como una primera aproximación al modelo de diseño (aunque sea un modelo en sí mismo), y es por tanto una entrada fundamental cuando se da forma al sistema en el diseño y en la implementación.

c) *Diseño*: Se modela el sistema y se encuentra la arquitectura para que soporte todos los requisitos. La entrada a esta fase es el modelo de análisis del flujo anterior e impone una estructura al sistema que debe conservarse en el futuro.

   Este flujo produce el modelo de diseño y el modelo de despliegue. El modelo de despliegue es un modelo de objetos que describe la distribución física del sistema incluyendo todos los nodos. Cada nodo representa un recurso de cómputo, normalmente un procesador. Los nodos poseen relaciones que representan medios de comunicación entre ellos, como Internet, Intranet, bus, etc.

d) *Implementación*: Se parte del resultado del diseño y se implementa el sistema en términos de componentes, es decir, ficheros de código fuente, *scripts*, ficheros de

código binario, ejecutables, etc. Los objetivos que se persiguen en este flujo pueden resumirse en:

- Planificar las integraciones de sistema necesarias en cada iteración. Se sigue para ello un enfoque incremental.
- Distribuir el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue.
- Implementar las clases y subsistemas encontrados durante el diseño.
- Probar los componentes individualmente, y a continuación integrarlos compilándolos y enlazándolos en uno o más ejecutables, antes de ser enviados para ser integrados y llevar a cabo las comprobaciones de sistema.

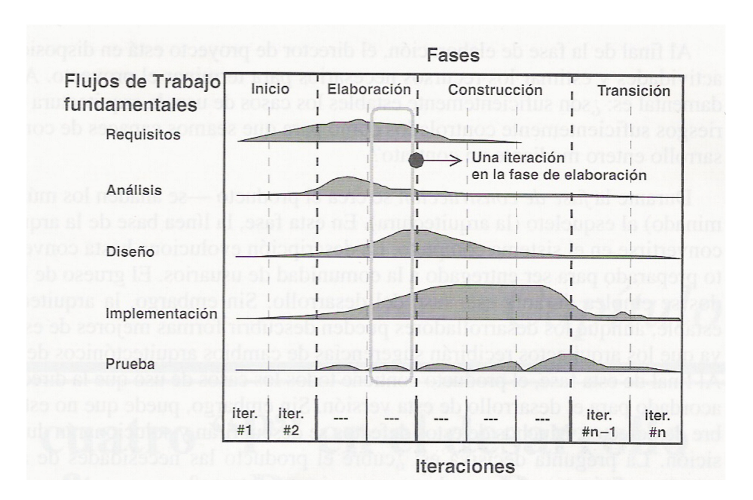


Figura 3.3. Incidencia de los flujos de trabajo en las fases de modelado.

e) *Prueba:* Se verifica el resultado de la implementación probando cada construcción, incluyendo tanto construcciones internas como intermedias, así como las versiones finales del sistema. Los objetivos de este flujo son:

- Planificar las pruebas necesarias en cada iteración, incluyendo las pruebas de integración y las pruebas de sistema. Las pruebas de integración son necesarias para cada construcción dentro de la iteración, mientras que las pruebas de sistema son necesarias sólo al final de la iteración.
- Diseñar e implementar las pruebas creando los casos de prueba que especifican qué probar, creando los procedimientos de prueba y si es posible, componentes de prueba ejecutables para automatizar las pruebas.
- Realizar las diferentes pruebas y manejar los resultados de cada prueba sistemáticamente. Las construcciones en las que se detectan defectos son

probadas de nuevo y posiblemente devueltas a otro flujo de trabajo, como diseño o implementación, de forma que los defectos puedan arreglarse.

La figura 3.3 [JACOBSON 00] muestra un resumen de la integración de los cinco flujos en las cuatro fases de cada ciclo en la vida del Proceso Unificado.

3.1.3.3.- *El Lenguaje Unificado de Modelado (UML).*

El Lenguaje Unificado de Modelado, en adelante UML, "*es un lenguaje gráfico para visualizar, especificar, construir y documentara los artefactos de un sistema con gran cantidad de software*" [BOOCH 01].

Este es el lenguaje ideal para ser utilizado en el Proceso Unificado de Desarrollo de Software como sus creadores afirman en [BOOCH 01] y [JACOBSON 00]. No se considera que sea objetivo del presente proyecto la exposición de las características de UML, por lo que únicamente se comentarán aquellos aspectos relevantes para el desarrollo del presente proyecto. En particular, y dado que se desarrolla una aplicación basada en Web, el apartado 3.2.5 describe los elementos del lenguaje que hacen referencia explícita al desarrollo de aplicaciones usando tecnologías basadas en Web.

3.2.- METODOLOGÍAS PARA EL DESARROLLO DE SITIOS WEB.

World Wide Web (WWW o Web), es un sistema basado en la hipermedia que proporciona la navegación por la información en Internet, usando hipervínculos (enlaces). Es la rama de Internet más conocida. Fue inicialmente desarrollada en 1992 y está formado por millones de páginas almacenadas en servidores de todo el mundo.

Desde su creación, el crecimiento de Internet y de WWW en particular ha sido espectacular. Comenzó estando formado por páginas compuestas por simple texto. Los avances en lenguajes y plataformas tecnológicas han permitido incluir en las páginas elementos mucho más complejos y técnicas multimedia.

A pesar de la enorme avalancha de aplicaciones y sistemas basados en Web, con muy diversos enfoques, objetivos, tamaños y complejidad, el desarrollo de los mismos suele realizarse, en muchos casos de manera específica, con falta de rigor y sistematización. Este desarrollo suele caracterizarse además por la total ausencia de técnicas o procedimientos que garanticen la integridad y el mantenimiento del sistema. En consecuencia, existe preocupación en lo que respecta al éxito tanto en el desarrollo como en el mantenimiento y estabilidad de aplicaciones a gran escala basadas en Web.

3.2.1.- Necesidad.

El desarrollo de aplicaciones Web lleva asociadas decisiones no triviales de diseño e implementación que inevitablemente influyen en todo el proceso de desarrollo. Los problemas involucrados, como el diseño del modelo del dominio y la construcción del interfaz de usuario, tienen requisitos disjuntos que deben tratarse por separado.

El alcance de la aplicación y el tipo de usuarios a los que estará dirigida son consideraciones tan importantes como las tecnologías elegidas para realizar la implementación. Así como las tecnologías pueden limitar la funcionalidad de la aplicación, decisiones de diseño equivocadas también pueden reducir su capacidad de extensión y reusabilidad. Es por ello que el uso de una metodología de diseño y de tecnologías que se adapten naturalmente a ésta, son de vital importancia para el desarrollo de aplicaciones complejas.

Existen tecnologías ampliamente usadas para el desarrollo de aplicaciones Web, pero muchas de ellas obligan a los desarrolladores a mezclar aspectos conceptuales y de presentación. Esto sucede principalmente con aquellas tecnologías no basadas en objetos. La elección de tecnologías complejas ralentiza el proceso de desarrollo y, en general, incrementa los costos, pero en ocasiones permite adecuarse a metodologías de diseño más fácilmente. Tal es el caso de las tecnologías orientadas a objetos, que tienden a demorar el desarrollo en las etapas iniciales.

El tiempo de desarrollo en la actualidad es crítico, tanto por razones de marketing como por límites en el presupuesto y los recursos, pero la adopción de estas tecnologías hace que el mantenimiento se transforme en una actividad más simple, la división en capas

sea tarea natural del desarrollo y el tiempo invertido en el diseño facilite el trabajo necesario para el resto de las actividades.

A continuación se realiza un resumen de las metodologías de desarrollo de aplicaciones Web más utilizadas, divididas en función de la técnica en que están basadas: en el modelo Entidad-Relación, en la orientación a objetos y basadas en UML. Se profundiza en sus metodologías abanderadas [KOCH 00]: RMM, OOHDM y WAE-UML.

3.2.2.- Basadas en el modelo Entidad-Relación.

3.2.2.1.- *RMM (Relationship Management Methodology).*

La metodología RMM ha sido ideada por Isakowitz, Stohr y Balasubramanian. Se basa en el modelo HDM[2]. Esta metodología es apropiada para dominios con estructuras regulares (es decir, con clases de objetos bien definidas, y con claras relaciones entre esas clases). Según sus autores, está orientada a problemas con datos volátiles, que cambian con mucha frecuencia, más que a entornos estáticos.

La base de la metodología es el modelo de datos RMDM *(Relationship Management Data Mode*l), que se genera a partir de un diagrama entidad-relación. Con él se describirá no sólo la información referente a las clases de objetos, sino también a la navegación entre ellos. Así, hay definidas unas primitivas para modelar los dominios (clases de objetos) y otras para el acceso a tales objetos. De entre las primeras, la más típica es la entidad. Como en la teoría relacional una entidad está compuesta por varios atributos. Además, en RMDM se incorpora una nueva primitiva muy importante denominada *slic*e, que define conjuntos de atributos de una entidad que se agrupan de forma lógica. Una entidad se representa mediante un recuadro y su nombre, mientras que un *slice* se representa mediante una figura similar a una gota de agua:

---

[2] HDM (Hypermedia Design Model) es un modelo de diseño de alto nivel. Surge con el objetivo de elaborar una metodología de desarrollo para sistemas hipermediales. Permite definir unas especificaciones de alto nivel del sistema a producir que pueden emplearse para realizar un diseño más detallado.

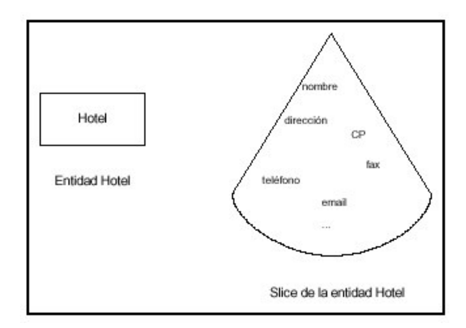


Figura 3.4. Slice en RMM.

En cuanto a las primitivas de acceso, se definen tres tipos de acceso diferentes. Para ver qué significa cada una se muestra un ejemplo. Suponiendo una entidad "Hotel" y que las instancias de esta entidad se pueden visitar según esos tres tipos de acceso:

- *Índice condicional*: en una pantalla aparecerá un índice alfabético de los nombres de los hoteles, y pulsando sobre uno, se podrá ver su información. Para ver los demás se debe volver al índice. En realidad el índice no es necesario que sea siempre una lista de las instancias como tal. Por ejemplo un índice para acceder a una entidad "ciudad" pudiera ser, por ejemplo, un mapa, donde al pulsar sobre una ciudad, se visitase su información.
- *Visita guiada condicional*: de la pantalla inicial se envía al primer hotel, y de él se puede pasar al segundo, y así sucesivamente, de forma que para llegar al último hay que pasar por todos los anteriores. Se permite volver a visitar al anterior hotel. En definitiva, es una estructura de acceso secuencial.
- *Visita guiada indexada condicional*: es una solución híbrida, donde se tiene un índice para acceder puntualmente a los elementos de la entidad, pero también se permite la navegación secuencial una vez seleccionado uno.
- Además, también se define una primitiva de grupo *(Groupin*g) mediante la cual se representa un menú. De una primitiva grupo colgarán tantas opciones como que haya en el menú.

La metodología está dividida en etapas. En la etapa 0, como toda metodología, comienza con un estudio de factibilidad y un análisis de los requerimientos (tanto de la información como de la navegación). También debe hacerse una selección del hardware y software que se necesitará. Los autores no hacen más referencias respecto de esta fase

inicial, por lo que es aplicable cualquier metodología típica de la ingeniería del software tradicional.

En la etapa 1 (Diseño Entidad-Relación) se confecciona un diagrama entidad-relación típico, desglosando las relaciones N:M en dos relaciones 1:N. El objetivo de esta fase es explicitar todos los enlaces entre objetos. Más tarde, las relaciones darán lugar a la navegación. Así, una relación especificará un camino en la navegación.

En la etapa 2 (Diseño de *slices*) se dividen las entidades en fragmentos significativos y se organizan en la red de navegación. Por ejemplo, toda la información de un hotel se colocaría en una ventana con *scroll* o bien en varias diferentes. Esta división se hace según la semántica de los atributos. Por ejemplo, se podría tener en un *slice* los datos generales del hotel y en otro un vídeo que muestre tomas del hotel. Cada *slice* agrupará uno o más atributos de una entidad, de tipos muy diferentes. Cada entidad tendrá su *head*, o *slice* principal, que se marca con un asterisco y que es *slice* al que, por defecto, se accede a través de los mecanismos de navegación. Entre los diferentes *slices* están los llamados enlaces estructurales, que nada tienen que ver con las relaciones, ya que al atravesar un enlace estructural, no se produce ningún cambio de contexto. Una vez determinados los *slices* hay que establecer los enlaces estructurales.

En la etapa 3 (Diseño navegacional) se sustituyen las relaciones por primitivas de acceso RMDM. En general, se preferirá una visita guiada a un índice cuando el número de instancias sea pequeño (menor de 10) y no exista un campo índice que pueda ayudar a los usuarios. Por el contrario, si el número es grande, se usarán índices.

Además en esta fase hay que elegir a qué *slice* se accede a través de una primitiva de acceso. Por defecto es el *slice* principal *(head*). En caso contrario, debe especificarse, etiquetando el nombre de la estructura de acceso. Por último, en esta fase se establece una jerarquía de menús, utilizando la primitiva de grupo o menú. Como regla general, se debe evitar grandes profundidades en la jerarquía, ya que desorientan al usuario. El resultado final de esta etapa es el diagrama RMDM.

Las etapas 4, 5, 6 y 7 (Interfaz de usuario y construcción) se realizan a partir del diagrama RMDM. En la Etapa 4 (Diseño de protocolos de conversión) se escriben unos

protocolos por los cuales se transforma cada elemento del RMDM en un objeto en la plataforma elegida.

En la Etapa 5 (Diseño de la interfaz de usuario) se diseñan gráficamente todas las pantallas correspondientes a cada uno de los *slices* que se han obtenido en la etapa 2.

En la Etapa 6 (Diseño del comportamiento en tiempo de ejecución) se decide qué mecanismos se utilizarán para guardar la historia, hacer *backtracking*s, permitir enlaces transversales, etc.

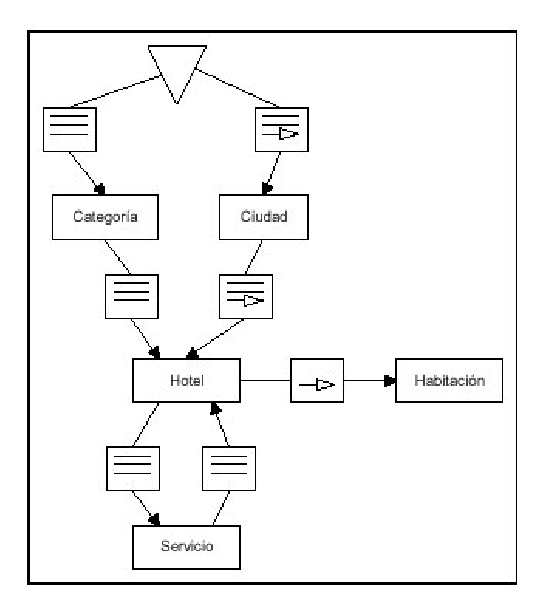


Figura 3.5. Diagrama RMDM.

En la Etapa 7 (Construcción y tests) se construye la aplicación y se realizan las pruebas testeando cuidadosamente todos los *paths* de la navegación.

3.2.2.2.- *RNA (Relationship-Navegational Analisys).*

Esta metodología propuesta por Bieber, Galgares y Lu en 1998, define una secuencia de pasos para aplicar en el desarrollo de aplicaciones Web centrándose especialmente en la fase de análisis. Está constituida por cinco fases. En la primera (fase de análisis de audiencia) se trata de identificar la audiencia de la aplicación. En la siguiente fase (análisis de elementos) se relacionan documentos, formularios, maquetas,

etc. de interés para la aplicación. La fase posterior (análisis de relaciones) trata de identificar esquemas, procesos y operaciones además de relaciones de diversos tipos. En la siguiente (análisis navegacional) se añaden las estructuras de navegación y por último en la fase de análisis de la implementación, el desarrollador decide cuales de las relaciones obtenidas en la tercera fase se van a implementar.

RNA propone únicamente unas guías para llevar a cabo las acciones de cada una de las fases; no se proponen conceptos de modelado ni notación alguna.

3.2.2.3.- *VHDM (View-based Hypermedia Design Methodology).*

Esta es una metodología orientada al diseño de aplicaciones hipermedia basada en vistas de usuario. Fue creada por Lee y otros en 1999. La metodología consiste en cinco fases: análisis de requisitos, modelado de datos, diseño de vistas, diseño de la navegación y elaboración. Las vistas se generan a partir del diagrama entidad-relación y son utilizadas en las dos últimas fases. Esta metodología resulta especialmente efectiva en la integración de bases de datos de empresas con sistemas hipermedia distribuidos en Internet.

3.2.3.- Basadas en técnicas orientadas a objetos.

3.2.3.1.- *EORM (Enhanced Object-Relationship Model).*

En esta metodología, propuesta por D. Lange, el proceso de desarrollo de un Sistema de Información Hipermedial (o Hiperdocumento) comprendería una primera fase de Análisis Orientado a Objetos del sistema, sin considerar los aspectos Hipermediales del mismo, obteniendo un Modelo de Objetos con la misma notación utilizada en OMT, que refleje la estructura de la información (mediante clases de objetos con atributos y relaciones entre las clases) y el comportamiento del sistema (a través de los métodos asociados a las clases de objetos) .

La idea fundamental de esta metodología es considerar un segunda fase, de Diseño, durante la cual se proceda a modificar el modelo de objetos obtenido durante el análisis añadiendo la semántica apropiada a las relaciones entre clases de objetos para convertirlas en enlaces Hipermedia, obteniendo finalmente un modelo enriquecido, que su autor denomina EORM, en el que se refleje tanto la estructura de la información (modelo abstracto Hipermedial compuesto de nodos y enlaces) como las posibilidades de navegación ofrecidas por el sistema. sobre dicha estructura, para lo cual existirá un repositorio o librería de clases de enlaces, donde se especifican las posibles operaciones asociadas a cada enlace de un hiperdocumento, que serán de tipo crear, eliminar, atravesar, siguiente, previo etc., así como sus posibles atributos (fecha de creación del enlace, estilo de presentación en pantalla, restricciones de acceso, etc.).

El desarrollo del sistema concluiría con una última fase de Construcción, durante la que se generaría el código fuente correspondiente a cada clase, y se prepararía su interfaz gráfico de usuario (GUI).

3.2.3.2.- *OOHDM (Object-Oriented Hypermedia Desing Method).*

Al igual que la metodología RMM, OOHDM (Metodología de Diseño Hipermedia Orientada a Objetos) está basado en HDM. Las principales diferencias son su concepción orientada a objetos y la inclusión de primitivas de modelado especiales para los diseños de navegación y de interfaz.

Las metodologías tradicionales de Ingeniería de Software, o las metodologías para sistemas de desarrollo de información, no contienen una buena abstracción capaz de facilitar la tarea de especificar aplicaciones Hipermedia. Una estructura de navegación robusta es una de las claves del éxito en las aplicaciones Hipermedia. Si el usuario entiende dónde puede ir y cómo llegar al lugar deseado, es una buena señal de que la aplicación ha sido bien diseñada. Construir la interfaz de una aplicación Web es también una tarea compleja; no sólo se necesita especificar cuáles son los objetos de la interfaz que deberían implementarse, sino también la manera en la cual estos objetos interactuarán con el resto de la aplicación.

En Hipermedia existen requisitos que deben satisfacerse en un entorno de desarrollo unificado. Por un lado, la navegación y el comportamiento funcional de la aplicación deben estar integrados. Por otro lado, durante el proceso de diseño se debe poder desacoplar las decisiones de diseño relacionadas con la estructura navegacional de la aplicación, de aquellas relacionadas con el modelo del dominio [DCAH].

OOHDM propone el desarrollo de aplicaciones Hipermedia a través de un proceso compuesto por cuatro etapas: diseño conceptual, diseño navegacional, diseño de interfaces abstractas e implementación.

a) *Diseño conceptual*. Durante esta actividad se construye un esquema conceptual representado por los objetos del dominio, las relaciones y colaboraciones existentes establecidas entre ellos. En las aplicaciones Hipermedia convencionales, cuyos componentes de Hipermedia no son modificados durante la ejecución, se podría usar un modelo de datos semántico estructural (como el modelo de entidades y relaciones). De este modo, en los casos en que la información base pueda cambiar dinámicamente o se intenten ejecutar cálculos complejos, se necesitará enriquecer el comportamiento del modelo de objetos. En OOHDM, el esquema conceptual está construido por clases, relaciones y subsistemas. Las clases son descritas como en los modelos orientados a objetos tradicionales. Sin embargo, los atributos pueden ser de múltiples tipos para representar perspectivas diferentes de las mismas entidades del mundo real.

   Se usa notación similar a UML y tarjetas de clases y relaciones similares a las tarjetas CRC (Clase - Responsabilidad - Colaboración). El esquema de las clases consiste en un conjunto de clases conectadas por relaciones. Los objetos son instancias de las clases. Las clases son usadas durante el diseño navegacional para derivar nodos, y las relaciones que son usadas para construir enlaces.

b) *Diseño navegacional.* La primera generación de aplicaciones Web se pensó para realizar navegación a través del espacio de información, utilizando un simple modelo de datos de Hipermedia. En OOHDM, la navegación se considera un paso crítico en el diseño aplicaciones. Un modelo navegacional se construye como una

vista sobre un diseño conceptual, admitiendo la construcción de modelos diferentes de acuerdo con los diferentes perfiles de usuarios. Cada modelo navegacional provee una vista subjetiva del diseño conceptual.

El diseño de navegación se expresa en dos esquemas: el esquema de clases navegacionales y el esquema de contextos navegacionales. En OOHDM existe un conjunto de tipos predefinidos de clases navegacionales: nodos, enlaces y estructuras de acceso. La semántica de los nodos y los enlaces son las tradicionales de las aplicaciones Hipermedia, y las estructuras de acceso, tales como índices o recorridos guiados, representan los posibles caminos de acceso a los nodos.

La principal estructura primitiva del espacio navegacional es la noción de contexto navegacional. Un contexto navegacional es un conjunto de nodos, enlaces, clases de contextos, y otros contextos navegacionales (contextos anidados). Pueden definirse por comprensión o extensión, o por enumeración de sus miembros. Los contextos navegacionales juegan un rol similar a las colecciones y fueron inspirados sobre el concepto de contextos anidados. Organizan el espacio navegacional en conjuntos convenientes que pueden ser recorridos en un orden particular y que deberían ser definidos como caminos para ayudar al usuario a lograr la tarea deseada. Los nodos se enriquecen con un conjunto de clases especiales que permiten de un nodo observar y presentar atributos (incluidos las anclas), así como métodos (comportamiento) cuando se navega en un particular contexto.

c) *Diseño de interfaz abstracta.* Una vez que se han definido las estructuras navegacionales, se deben especificar los aspectos de interfaz. Esto significa definir la forma en la cual los objetos navegacionales pueden aparecer, cómo los objetos de interfaz activarán la navegación y el resto de la funcionalidad de la aplicación, qué transformaciones de la interfaz son pertinentes y cuándo es necesario realizarlas.

Una clara separación entre diseño navegacional y diseño de interfaz abstracta permite construir diferentes interfaces para el mismo modelo navegacional, dejando un alto grado de independencia de la tecnología de interfaz de usuario. El aspecto de la interfaz de usuario de aplicaciones interactivas (en particular las aplicaciones Web) es un punto crítico en el desarrollo que las modernas metodologías tienden a descuidar. En OOHDM se utiliza el diseño de interfaz abstracta para describir la interfaz del usuario de la aplicación de Hipermedia.

El modelo de interfaz ADVs (Vista de Datos Abstracta) especifica la organización y comportamiento de la interfaz, pero la apariencia física real o de los atributos, y la disposición de las propiedades de las ADVs en la pantalla real se hacen en la fase de implementación.

d) *Implementación.* En esta fase, el diseñador debe implementar el diseño. Hasta ahora, todos los modelos se han construido de forma independiente de la plataforma de implementación; en esta fase se tiene en cuenta el entorno particular en el cual se va a ejecutar la aplicación.

Al llegar a esta fase, el primer paso que debe realizar el diseñador es definir los ítems de información que son parte del dominio del problema. Debe identificar también, cómo son organizados los ítems de acuerdo con el perfil del usuario y su tarea; decidir qué interfaz debería ver y cómo debería comportarse. A fin de implementar todo en un entorno Web, el diseñador debe decidir además qué información debe almacenarse.

3.2.3.3.- *WSDM (Web Site Design Method).*

Esta metodología está centrada en el usuario. WSDM (Metodología de Diseño de Sitios Web) define la información en función de los requerimientos de los usuarios de una aplicación Web. Ha sido propuesta por De Troyer y Leune (1997) y tiene 3 fases: modelo de usuario, diseño conceptual y diseño de la implementación.

En la primera fase los usuarios/visitantes potenciales de la Web se identifican y clasifican, por ejemplo, en función de sus intereses y preferencias de navegación. El diseño conceptual se divide en dos subfases: modelo de objetos y diseño navegacional. A su vez el modelo de objetos se divide en: modelo de objetos de negocio, de usuario y de perspectivas.

3.2.3.4.- *SOHDM (Scenario-based Object-oriented Hypermedia Design Methodology).*

Otra metodología desarrollada por Lee y Yoo en 1998 es SOHDM (Metodología de Diseño Hipermedia orientada a Objetos basada en eScenarios). Comprende seis fases:

análisis de dominio, modelo de objetos, diseño de vistas, diseño navegacional, diseño de la implementación y construcción de la implementación. Tiene conceptos similares a RMM, OOHDM y EORM. Difiere en el uso de escenarios, los cuales se describen mediante gráficos de actividad basado en eventos.

Los escenarios se definen en la fase de análisis de dominio y se usan para el modelo de objetos. El diseño de vistas consiste en determinar vistas orientadas a objetos generadas a partir de las clases o sus asociaciones. El diseño navegacional usa los escenarios para determinar el acceso a los nodos de la estructura.

3.2.4.- Basadas en UML.

3.2.4.1.- *UWE (UML-based Web Engeneering).*

UWE es una propuesta metodológica basada en el Modelo de Proceso Unificado y en el lenguaje UML para el desarrollo de aplicaciones Web. Cubre todo el ciclo de vida, proponiendo un método orientado a objetos iterativo e incremental. Utiliza la notación y los diagramas UML para el análisis y diseño de la aplicación Web. Los aspectos especiales de la Web se modelan definiendo un perfil llamado *lightweight*.

El perfil de UWE incluye estereotipos y valores etiquetados definidos para modelar los elementos necesarios para el modelado de la navegación o de la presentación de las aplicaciones Web. Para cada aspecto se construye un modelo siguiendo las pautas UWE.

El *modelo de navegación* se construye a partir del modelo conceptual en dos pasos, crea el modelo del espacio de navegación (especifica qué objetos pueden visitarse y cómo se alcanzan) y después crea el modelo de la estructura de la navegación (cómo realizar la navegación utilizando elementos de acceso como índices, menús, etc. Estos dos modelos y los caminos de navegación forman el *modelo de estructura navegacional.*

Finalmente, el modelo de presentación modelos los elementos de la presentación que se usarán en las clases navegacionales y los elementos de acceso.

3.2.4.2.- *OO-H (Object Oriented Hypermedia).*

OO-H es una extensión a los métodos orientados a objetos para el modelado y generación automática de interfaces hipermediales. Sus autores [CACHERO 03] señalan que es un método genérico basada en el paradigma orientado a objetos que proporciona al diseñador la semántica y la notación necesaria para el modelado de interfaces basados en Web, definiendo un conjunto de diagramas, técnicas y herramientas que se integran en un proceso de diseño flexible.

Uno de los principales objetivos de OO-H es integrar constructores, modelos y prácticas relevantes del marco de las aproximaciones del campo hipermedia, en una propuesta dentro del proceso de desarrollo adaptado a la naturaleza de los interfaces Web. Los autores, también afirman que OO-H es útil para el modelado y generación completa de aplicaciones Web.

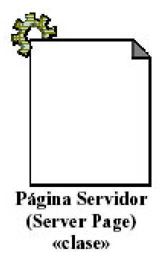
OO-H pretende mejorar las capacidades para el desarrollo de interfaces hipermediales de calidad. Puede considerarse dirigida por los requisitos de usuario. No pretende cubrir todo el ciclo de vida de las aplicaciones Web, sólo se centra en parte de las actividades e intenta integrar el proceso de desarrollo dentro de un marco en espiral que refleje el carácter iterativo e incremental del método.

Existe una herramienta llamada VisualWADE que soporta OO-H. Es una herramienta CASE que automatiza la producción de interfaces de usuario de aplicaciones Web. Es una aproximación dirigida por el modelo que pone énfasis en el análisis de alto nivel y prototipado rápido [GOMEZ 03].

3.2.5.- Lenguaje de Modelado WAE-UML (Web-Application Extension for UML).

WAE propone todos los pasos a seguir para la elaboración de un sitio Web incidiendo en el diseño. El autor, Jim Conallen, realiza una extensión del lenguaje UML que permite modelar los elementos que intervienen en el desarrollo de aplicaciones Web, facilitando la labor de diseño y mejorando la visión global del sistema. Dicha extensión define un conjunto de estereotipos (de clase, de asociación, de atributos y de

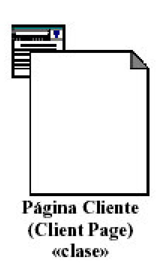
componentes), valores etiquetados y restricciones. A continuación se muestran los elementos del modelo [CONALLEN 02]:



Una página de servidor representa una página Web que tiene *scripts* que son ejecutados por el servidor. Estos *scripts* interactúan con recursos del servidor como bases de datos, lógica de negocio y sistemas externos. Las operaciones del objeto representan las funciones en el *script*, y sus atributos representan las variables que son visibles en el ámbito de la página (accesibles por todas las funciones de la página).

Tiene la restricción de que sólo puede tener relaciones con objetos localizados en el servidor.

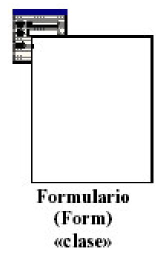
Valores etiquetados: el motor del *script*: Tanto el lenguaje como el motor que se debería usarse para ejecutar o interpretar esta página (PHP, ASP, Perl, etc.).



Una instancia de una página cliente es una página Web con formato HTML y es una mezcla de datos, presentación e incluso lógica. Las páginas clientes son representadas por los navegadores clientes, y pueden contener *scripts* que son interpretados por el navegador. Las funciones de la página cliente mapean las funciones en las etiquetas *script* de la página. Los atributos de la página cliente mapean las variables declaradas en las etiquetas *script* de la página, que son accesibles por una función en la página (con ámbito de página). Las páginas cliente pueden tener asociaciones con otras páginas cliente o servidor.

No tiene restricciones:

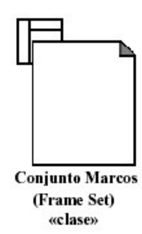
Valores etiquetados: EtiquetaTítulo (`TitleTag`): el título de la página lo muestra el Navegador EtiquetaBase (`BaseTag`): la URL base por referencia de URLs relativos. EtiquetaCuepo (`BodyTag`): el conjunto de atributos para la etiqueta `<Body>` que establecen el color de fondo y los atributos del texto por defecto.



Una clase estereotipada como «form» es una colección de campos de entrada que forman parte de una página cliente. Una clase form se mapea directamente con la etiqueta HTML `<form>`. Los atributos de esta clase representan los campos de entrada del formulario HTML (input boxes, text areas, radio buttons, check boxes, y campos hidden). Un «form» no tiene operaciones, por lo que no pueden encapsularse en un formulario. Cualquier operación que interactúe con el formulario es una propiedad de la página que contiene al formulario.

No tiene restricciones.

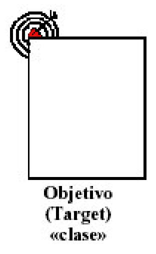
Valores etiquetados: Método (`Method`) – el método usado para enviar datos al URL del action, puede tomar los valores `GET` o `POST`.



Es un contenedor de múltiples páginas Web. La zona de visualización de la Web se divide en marcos rectangulares. Cada marco se asocia con un único objetivo (`target`). El contenido de un marco puede ser una página u otro marco.

No tiene restricciones.

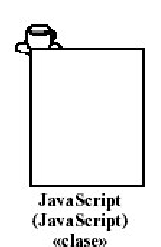
Valores etiquetados: Filas – con el número de píxels o el porcentaje de ventana con la altura de las filas. Columnas - con el valor del número de píxels o el porcentaje de ventana con la anchura de las columnas.



Es la zona de la ventana del navegador donde se visualizarán las páginas Web. El nombre de la clase estereotipada es el nombre del objetivo.

Restricciones: El nombre de un objetivo debe ser único para cada cliente del sistema.

No tiene valores etiquetados.



Define objetos del usuario. Sólo es posible en navegadores que soporten JavaScript. Sólo existen en el contexto de páginas cliente.

No tiene restricciones.

No tiene valores etiquetados.

Enlace (Link) «asociación»

Un link (enlace) es un puntero desde una página cliente a otra página. En un diagrama de clases, un enlace es una asociación entre una página cliente y cualquier otra página cliente o una página servidor. Una asociación Link se mapea directamente con la etiqueta HTML ancla `<a>`.

No tiene restricciones.

Valores etiquetados: Parámetros: una lista de nombres de parámetros que deberían pasarse con la petición de la página enlazada.

Enlace Objetivo (Targeted Link) «asociación»

Es similar a la asociación de enlace. Es un enlace donde la página asociada es visualizada en otro objetivo (`target`). Se corresponde con `<a ref.=”...” target=”....”>`.

No tiene restricciones.

Valores etiquetados: Parámetros: los parámetros pasados a la página enlazada. Nombre del objetivo: nombre del objetivo donde se visualizará la página.

Contenido de Marco (Frame Content) «asociación»

Es una asociación de agregación que indica el contenido del marco de otra página u objetivo.

No tiene restricciones.

Valores etiquetados: Fila y Columna.

Presentar/Enviar (Submit) «asociación»

Una asociación “submit” es siempre entre un formulario y una página servidor. Los formularios envían los valores de sus campos al servidor a través de páginas servidor para procesarlos. El servidor Web procesa la página servidor, la cual acepta y usa la información dentro del formulario enviado.

No tiene restricciones.

Valores etiquetados: Parámetros: una lista de nombres de parámetros que deberían ser pasados con la petición de la página enlazada.

Construcciones (Builds) «asociación»

La asociación «builds» es un tipo especial de relación que une el vacío entre las páginas cliente y de servidor. Las páginas de servidor existen únicamente en el servidor. Son usadas para crear páginas cliente.

La asociación «builds» identifica que página de servidor es responsable de la creación de una página cliente. Ésta es una relación direccional, pues la página cliente no tiene conocimiento de cómo ha sido creada. Una página de servidor puede crear múltiples páginas cliente, pero una página cliente tan solo puede ser construida por una página de servidor.

No tiene restricciones.

No tiene valores etiquetados.

Redirección (Redirect) «asociación»

Una asociación «redirect» es una asociación unidireccional con otra página Web. Puede ser dirigida desde y hasta una página cliente o de servidor. Si la relación se origina desde una página servidor entonces indica que el procesado de la página solicitada debe continuar en la otra página. Esto indica que la página destino siempre participa en la creación de la página cliente. Esta relación no es completamente estructural, pues la invocación de una operación de redirección se debe hacer a través de programación en el código de la página origen. Si la relación se origina desde una página cliente entonces esto indica que la página destino será automáticamente solicitada por el navegador, sin la participación activa del usuario. Se puede especificar un tiempo de demora (en segundos) antes de que la segunda página sea solicitada. El uso de la redirección se corresponde con la etiqueta `META` y `HTTP-EQUIV` el valor de `Refresh`.

No tiene restricciones.

Valores etiquetados: Demora - Cantidad de tiempo que una página cliente debería esperar para ser redirigida a la siguiente página. Este valor se corresponde con el atributo `Content` de la etiqueta `Meta`.

Objeto (Object) «asociación»

Asociación entre una página de cliente y un objeto que está embebido en ella (un applet Java, un ActiveX, etc. Se corresponde con la etiqueta `<object>` de HTML.

No tiene restricciones.

Valores etiquetados: Parámetros: lista de parámetros separados por punto y coma seguida de una serie de valores opcionales.

IIOP
(IIOP)
«asociación»

Es un tipo de relación especial entre objetos cliente y objetos del servidor. Representa mecanismos de comunicación (por ejemplo, Java Beans en el cliente y Java Beans en el servidor).

No tiene restricciones ni valores etiquetados.

RMI
(RMI)
«asociación»

Es un mecanismo para enviar mensajes entre Applets y Beans de Java entre máquinas distintas.

No tiene restricciones ni valores etiquetados.

Elemento de Entrada
(Input Element)
«atributo»

Un Input Element es un atributo de un objeto «Form». Se mapea directamente con la etiqueta HTML `<input>`. Este atributo es usado para introducir una palabra o una línea de texto. Los valores etiquetados asociados con este atributo estereotipado se corresponden con los atributos de la etiqueta `<input>`. Para completar los valores requeridos por la etiqueta HTML; el nombre del atributo se usa como el nombre de la etiqueta `<input>`, y el valor inicial del atributo es el valor de la etiqueta.

No tiene restricciones.

Valores etiquetados: `Type: e`l tipo del control input puede ser (`Text, Number, Password, Checkbox, Radio, Submit, Reset). Size` – Especifica la longitud del área que aparece en pantalla, en caracteres. `Maxlength` – El máximo número de caracteres que el usuario puede introducir.

Elemento de Selección
(Select Element)
«atributo»

Es un control input usado en los formularios. Este control permite al usuario seleccionar uno o más elementos de una lista.

No tiene restricciones.

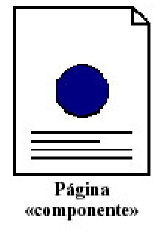
Valores etiquetados: `Size: e`specifica cuantos campos se muestran al mismo tiempo. `Multiple: b`ooleano que indica puede ser seleccionados múltiples campos de la lista.

Elemento de Área de Texto
(TextArea Element)
«atributo»

Es un control input usado en los formularios que permite introducir múltiples líneas.

No tiene restricciones.

Valores etiquetados: `Rows`: número de filas de texto visibles. `Cols`: el ancho visible del control especificado por el valor medio del ancho del carácter.



Un componente página es una página Web. Puede ser solicitada por su nombre por un navegador. Un componente página puede contener o no *scripts* cliente o servidor. Típicamente los componentes página son ficheros de texto accesibles por el servidor Web, pero también pueden ser módulos compilables que son cargados e invocados por el servidor Web. Finalmente cuando se accede a través del servidor Web una página produce un documento con formato HTML que se envía como respuesta a la petición de un navegador.

No tiene restricciones.

Valores etiquetados: `Path`: la ruta requerida para especificar la página Web en el servidor Web. Este valor debería ser relativa al directorio raíz de la aplicación Web.

Para cada estereotipo, se permiten las combinaciones de la Figura 3.6.

	Página Cliente	Página Servidor	Cjto. Marcos	Objetivo	Formulario
Página Cliente	Enlace Enlace Objetivo Redirección	Enlace Enlace Objetivo Redirección	Enlace Enlace Objetivo Redirección	Dependencia	Agregación
Página Servidor	Construcciones Redirección	Redirección	Construcciones Redirección		
Cjto. Marcos	Contenido Marco		Contenido Marco	Contenido Marco	
Objetivo					
Formulario	"Agregado por"	Presentar			

Figura 3.6. Combinaciones de estereotipos permitidas con WAE.

3.3.- ARQUITECTURA DE APLICACIONES WEB.

Tradicionalmente las aplicaciones cliente-servidor estaban basadas en una arquitectura de dos capas que permitía la separación de la parte cliente de la parte servidora. La necesidad de escalabilidad hace que se implante una nueva arquitectura para el desarrollo de aplicaciones Web, ya que la arquitectura de dos capas presenta problemas

importantes. Por ejemplo, el cliente debe tiene demasiado peso y requiere ordenadores potentes. Además, el cliente es el que maneja la lógica de la aplicación en su máquina, con todos los inconvenientes que presenta (aplicaciones modificadas por algunos usuarios maliciosos, problemas para la escalabilidad con nuevas versiones de la aplicación, ...).

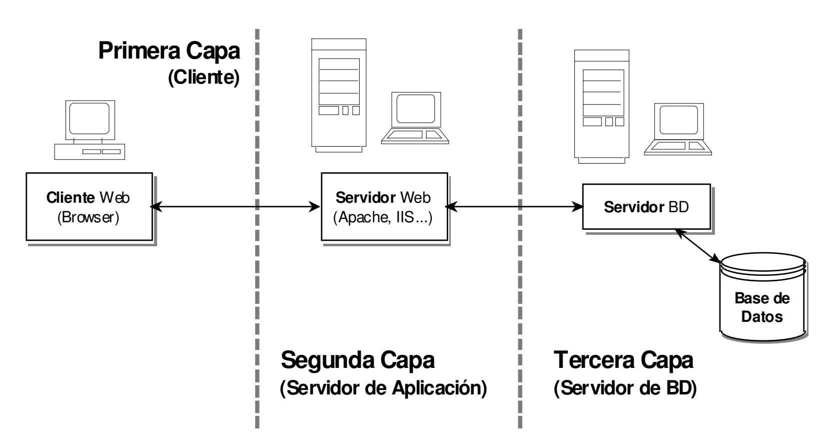


Figura 3.7. Arquitectura de tres capas.

En 1995 se plantea una variación de la típica arquitectura de dos capas que soluciona estos problemas y mejora la escalabilidad de las aplicaciones. Esta nueva arquitectura, puede apreciarse en la Figura 3.7 [GONZALEZ 02], posee tres capas.

En la primera capa, el cliente es ahora responsable únicamente del interfaz de usuario y posiblemente de una pequeña lógica del negocio (que nos permita simplificar la lógica de control a través de la red a la vez que disminuir la saturación de la misma). Esta pequeña lógica suele limitarse a validar las entradas del cliente.

La segunda capa, conectada a la vez con el cliente y con la capa de acceso a bases de datos, se encarga ahora del control de toda la lógica de la aplicación, así como de procesar los datos que utiliza la misma. El servidor de aplicación debe estar optimizado para permitir la conexión a un número alto de clientes simultáneamente.

La tercera capa se encarga de la validación final de los datos y el acceso a la base de datos para guardar los resultados del procesamiento de la capa anterior.

Este enfoque en tres capas tiene muchas ventajas respecto del enfoque tradicional de dos capas. Las más importantes son: un pequeño cliente, el mantenimiento de la aplicación está centralizado y modularidad en las capas.

3.3.1.- Patrones de arquitectura para aplicaciones Web.

Las arquitecturas para aplicaciones Web son muy diversas, pero todas incluyen un navegador para el cliente, un servidor Web, un servidor de aplicaciones y el uso de una base de datos.

En [CONALLEN 02] se definen tres patrones de arquitectura que pueden utilizarse separadamente o combinarse para el desarrollo de una aplicación Web:

a) *Thin Web Clients (Clientes Web Ligeros).* Se caracteriza por que el cliente sólo requiere un navegador estándar, tiene muy poca parte de control y la lógica de funcionamiento está en el servidor. Es el más indicado para la construcción de páginas dinámicas de servidor, ya que el cliente solo tiene capacidades de presentación y toda la lógica de negocio está en el lado del servidor. Los principales componentes son:
    - Navegador: funciona como interfaz para la aplicación.
    - Servidor Web: único punto de acceso para los clientes; dependiendo de las peticiones, puede activar determinados procesos. El resultado siempre es una página HTML.
    - HTTP: es el protocolo de comunicación entre navegador y servidor "sin conexión" que se usará de forma más común. Puede usarse HTTPS.
    - Páginas HTML: contendrá el interfaz de usuario y la información que se le quiera presentar. No necesita procesamiento por parte del servidor.
    - Páginas de servidor (páginas dinámicas): tienen acceso a los recursos del servidor, y contienen información que debe ser procesada antes de enviarse.
    - Servidor de la aplicación: el principal motor de funcionamiento. Es responsable de ejecutar el código de las páginas de servidor.
    - Servidor de base de datos: es la parte del sistema que mantiene la persistencia del sistema.
    - Sistema de ficheros: en buena parte de los casos su estructura de directorios determina la URL de las páginas. Es responsable de los

ficheros de páginas estáticas (para el servidor Web) y las dinámicas (para el servidor de aplicaciones).

b) *Thick Web Clients (Clientes Web Pesados).* Sus características se basan en el dinamismo por parte del cliente. Parte de la lógica de funcionamiento se encuentra en el cliente. El cliente usa HTML dinámico (*scripts*), applets o controles activeX, etc. Esta arquitectura extiende los clientes Web ligeros con el uso de objetos que se ejecutan en el cliente. Esta arquitectura es más apropiada cuando se puede suponer que el usuario tendrá ciertas versiones de un navegador y con una configuración determinada, cuando se quiere una interfaz de usuario elaborada o cuando parte de la lógica de la aplicación se puede delegar en el cliente. Toda la comunicación entre cliente y servidor se realiza a través de HTTP. Además de los componentes de los *Thin Web Clients*, se añaden:

- *Scripts* de cliente: son interpretados por el navegador.
- Documentos XML.
- Controles ActiveX: Código binario ejecutable que se añade al navegador vía HTTP.
- Applets: se usan para potenciar las interfaces, analizar documentos XML o implementar comportamientos complejos de la aplicación.
- JavaBeans.
- ....

c) *Web Delivery (Distribución Web)*. Se caracterizan por soportar otros protocolos, además de HTTP, que permiten incluir objetos distribuidos. El navegador funciona como un interfaz para un sistema de objetos distribuidos. Acaba siendo una aplicación cliente-servidor con objetos distribuidos que usa un navegador y un servidor Web como entrada de dichos objetos. Este tipo de aplicación es apropiada cuando es necesario un alto grado de control sobre el cliente y la configuración de la red.

No es una arquitectura muy adecuada para aplicaciones que funcionen en Internet, o en redes poco fiables, siendo utilizada normalmente en Intranets, dada su seguridad y rapidez. Esta arquitectura se usa en sistemas complejos en

los que gran parte está implementado como un cliente Web pesado, y una parte de la aplicación, de acceso restringido, sigue esta arquitectura.

La diferencia más significativa con las arquitecturas anteriores se encuentra en el método de comunicación cliente-servidor y sus elementos más significativos que incluye son DCOM, IIOP o RMI (JRMP). Con la comunicación directa y persistente entre cliente y servidor, se evitan las limitaciones de las anteriores arquitecturas. El cliente puede realizar lógica del negocio importante hasta un grado mucho mayor.

La mayor desventaja de esta tecnología es que la portabilidad disminuye notablemente, también requiere una red fiable y una caída del servidor puede ser muy problemática.

La Figura 3.8 muestra un diagrama que resume el planteamiento a tener en cuenta para elegir el patrón de arquitectura adecuado.

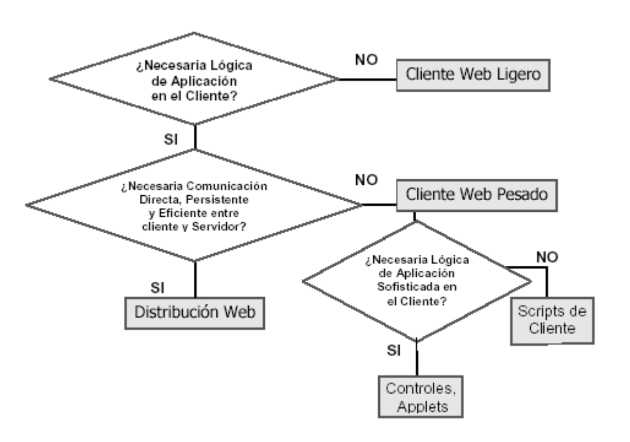


Figura 3.8. Elección del patrón de arquitectura adecuado.

3.4.- PRINCIPIOS DE DISEÑO DE SITIOS WEB.

Todo el mundo está de acuerdo en que el desarrollo de sitios Web necesita talento y equilibrio en la forma, la función, el contenido y la finalidad. En cambio, no es fácil definir exactamente qué se entiende por un buen diseño Web. El diseño está condicionado por el objetivo del sitio, el estilo de la compañía, el contenido, la audiencia, la rentabilidad económica, etc.

El diseño Web puede definirse como "*una búsqueda multidisciplinar centrado en el usuario que incluye influencias de las artes visuales, la tecnología, el contenido y la finalidad*" [POWELL 01]. A continuación se enumeran algunos principios básicos de diseño Web.

a) *Escritura de contenidos para la red*. Cada página no debe contener mucho texto. Los usuarios no leen, echan un vistazo en busca de la información que les interesa. Cada elemento que se presenta compite con el resto, por lo que se debe evitar incluir información superflua. La información del texto es conveniente que tenga las siguientes características: texto estructurado, inclusión de sumarios, índices de contenido, títulos y subtítulos claros y concisos, un párrafo-una idea, lenguaje objetivo y poco técnico, colores discretos que faciliten la fatiga ocular, etc.

b) *Imágenes*. Se deben incluir las imprescindibles ya que distraen la atención del usuario y les dificulta centrarse en el contenido de la página. Deben tener tamaño reducido para reducir el tiempo de carga. Todas deben utilizar el atributo `<IMG ALT="..">` para facilitar la navegación de los usuarios y para favorecer la accesibilidad de los discapacitados.

c) *La navegación*. La navegación es un medio para conseguir un fin [POWELL 01]. Todas las páginas necesitan tener un área de navegación en la parte superior, que indiquen la situación de la página en el sitio y que permitan al usuario navegar por el sitio. El área de navegación no debe contener un número excesivo de enlaces. Tampoco conviene que la jerarquización de la estructura sea demasiado profunda. Resulta muy interesente incluir un mapa del sitio para facilitar una visión global del sitio. Los enlaces deben ser homogéneos y se recomienda que el título del enlace informe sobre lo que se encontrará en el destino.

d) *La página inicial*. Es la primera que se visualiza y debido a la lentitud de la red, es fundamental. El orden temporal de visualización debe estar jerarquizado de mayor a menor utilidad y relevancia. Para reducir subjetivamente el tiempo de espera de los usuarios se puede utilizar un indicador de tiempo de espera o un indicador del

rendimiento del sistema; y si la espera será larga, intentar entretener al usuario con otros elementos atractivos.

e) *La búsqueda de información*. Los buscadores deben ser capaces de encontrar la información a pesar de pequeños errores en la escritura de la palabra. Cuando el volumen de documentos encontrados pueda ser grande, es necesario incluir la opción de búsqueda en todas las páginas del sitio. El buscador debe buscar en todo el sitio y las búsquedas parciales o booleanas se deben restringir a una sección de búsqueda avanzada para no confundir a los usuarios sin experiencia.

f) *Seguridad para usuarios*. En los sitios en los que exista información privada mediante registro de usuario, se debe reducir la dificultad en el registro. El uso de cookies facilita esta tarea. Es aconsejable que los usuarios puedan elegir su nombre de usuario y cambiar su contraseña.

A continuación se muestran brevemente otros principios básicos de diseño Web [POWELL 01] y [NIELSEN 00]:

- Anticipación a las necesidades de los usuarios.
- Los usuarios deben tener control de la aplicación.
- Utilizar hojas de estilos para homogeneizar el aspecto del sitio.
- Usar los colores con precaución.
- Buscar la productividad del usuario, no de la máquina.
- Permitir la reversibilidad de acciones.
- Reducir el tiempo de latencia.
- Mínimo proceso de aprendizaje.
- Legibilidad e intuitividad.
- Incluir ayuda integrada.
- Facilitar acceso rápido a los enlaces más consultados (favoritos).
- Minimizar el uso de alta tecnología (imposibilita el acceso a usuarios, ralentiza la visualización, etc).

3.5.- TECNOLOGÍAS PARA EL DESARROLLO DE APLICACIONES WEB.

En este apartado se estudian las tecnologías más importantes y utilizadas que actualmente existen en el mercado para el desarrollo de aplicaciones Web, partiendo del lenguaje HTML que fue la primera herramienta que permitió diseñar y desarrollar páginas Web tras el desarrollo de World Wide Web para Internet.

3.5.1.- HTML (HyperText Markup Language).

HTML es un lenguaje de formato de hipertexto de propósito general basado en SGML[3]. Especifica un formato de los documentos que permite órdenes para el formato del texto, así como órdenes para enlaces de hipertexto y órdenes para la exhibición de imágenes. HTML es un lenguaje estándar, simple, potente e independiente de la plataforma.

Es el lenguaje más antiguo de desarrollo de páginas Web. Únicamente permite desarrollar páginas Web estáticas, lo que, evidentemente, lo convierte en poco eficaz ante las necesidades actuales. No obstante, y tras numerosas revisiones y actualizaciones sigue siendo el motor de la interacción servidor de Internet – navegador. De hecho, el resto de tecnologías (CGI, lenguajes de *script*, sistemas gestores de contenidos, etc.) siguen traduciendo su código, de forma que al navegador le retornan HTML estándar.

3.5.2.- CGI (Common Gateway Interface).

Esta tecnología es la primera que permite desarrollar páginas Web dinámicas. Una página Web dinámica es aquella que se genera en el servidor ante una petición de un cliente, mientras que una página Web estática está almacenada de forma fija en el servidor HTTP sirviéndose tal cual cuando es solicitada.

---

[3] SGML (Lenguaje eStándar de Marcas Generalizado) se desarrolla para cubrir el hueco existente entre el texto sin formato y los lenguajes para la descripción de las páginas o para el formato del texto [GONZALEZ 02].

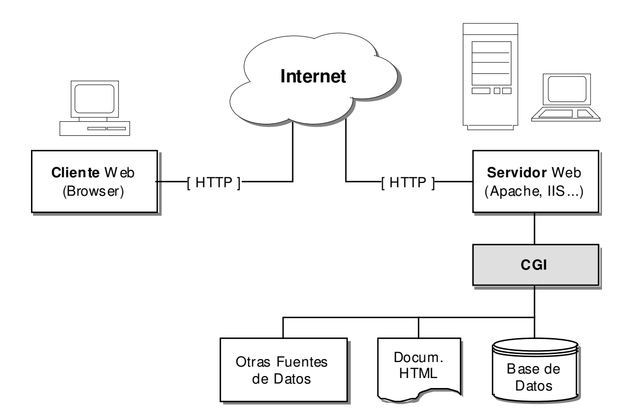


Figura 3.9. Funcionamiento de página Web dinámica con CGI.

CGI es una especificación para que un servidor Web y un programa puedan comunicarse. Define cómo los programas (*scripts*) se comunican con los servidores Web. Un programa CGI tiene que aceptar y devolver datos conforme a este protocolo. Recogerá los datos de la entrada estándar (normalmente, se reciben a través del envío de formularios codificados en HTML) y escribirá en la salida estándar Además, el programa tiene que enviar la cabecera de tipo `MIME` antes del cuerpo de los datos (`Content-type`: [tipo], donde tipo puede ser `text/html, text/plain, image/jpeg`, etc.).

Los programas CGI pueden escribirse en cualquier lenguaje. Algunos de los más utilizados son Perl, Python, C, C++, Visual Basic, etc.

3.5.3.- Lenguajes de *script*.

Los denominados lenguajes de *script* del lado del servidor son un método de desarrollo que permiten la generación de páginas Web dinámicas creadas en el servidor. A diferencia de CGI, tanto PHP (Hypertext PreProcessor) como ASP (Active Server Pages), los lenguajes más populares y usados, se ejecutan en el mismo proceso que el servidor, y la posibilidad de utilización de múltiples hilos, hacen que sea la solución elegida para servidores que tendrán que soportar un gran volumen de usuarios.

Cuando un cliente solicita una página cuya extensión indica que es dinámica, el servidor que atiende la petición buscando el fichero asociado y se lo entrega al motor del

lenguaje. Éste analiza el fichero, accediendo a bases de datos si fuera necesario, y entrega un resultado HTML puro al servidor que únicamente hace de puente entregando el fichero directamente entendible por el navegador del cliente. Esta solución, al igual que la basada en CGI, es totalmente independiente del navegador y del sistema operativo que utilice el cliente.

Desde el punto de vista funcional, con ambos lenguajes se puede hacer prácticamente lo mismo, pero existen ciertas diferencias muy importantes a la hora de desarrollar con una u otra plataforma. Para empezar, PHP es software libre (disponibilidad de código fuente, además de sin necesidad de adquirir licencia de uso...). ASP es un lenguaje propietario y para realizar algunas operaciones avanzadas (encriptación de la comunicación para acceder a una base de datos segura), es necesario adquirir las licencias de los módulos adicionales.

Tanto PHP como ASP permiten que el código propio del lenguaje se inserte en el código HTML. Esto es muy cómodo a la hora de programar porque basta con insertar el código dinámico entre el esqueleto HTML que se haya realizado antes.

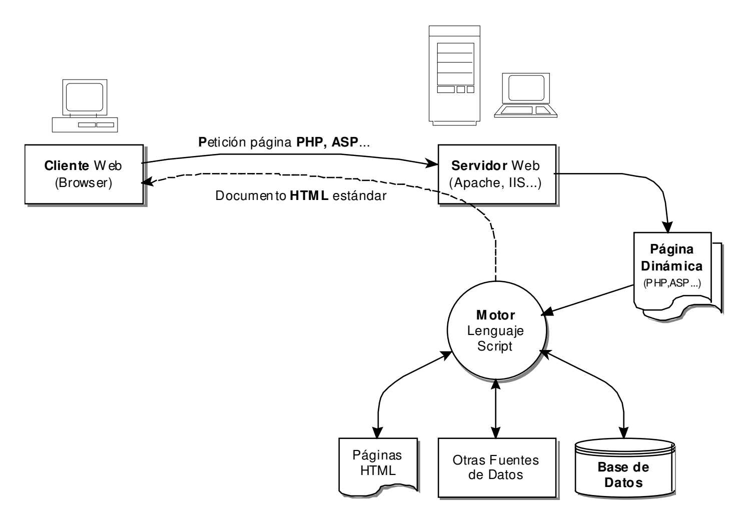


Figura 3.10. Funcionamiento de los lenguajes de *script* del lado del servidor.

Como puede apreciarse en la Figura 3.10 [GONZALEZ 02], los lenguajes de *script* tienen motores de acceso a bases de datos (generalmente relacionales), incluso soportan ODBC para la integración con cualquier Sistema Gestor de Bases de Datos. Debe destacarse que la mayoría de desarrollos Web con este tipo de lenguajes están montados sobre una base de datos relacional (MS Access, SQL Server, Oracle, Mysql, etc.).

3.5.4.- <u>Basadas en Java.</u>

Java es un lenguaje de programación orientado a objetos desarrollado por la compañía Sun Microsystems. Está construido a partir de lenguajes orientados a objetos anteriores, como C++, pero no pretende ser compatible con ellos sino ir mucho más lejos, añadiendo nuevas características como recolección de basura, programación multihilo y manejo de memoria a cargo del lenguaje.

JavaScript es un lenguaje de *script* basado en Java. El código se ejecuta en el cliente. Este lenguaje, se utiliza para realizar detalles de la capa de presentación de las aplicaciones integrado con el HTML estándar. En ningún caso, es un lenguaje aconsejable para desarrollar sitios Web con un volumen considerable de información que necesiten acceso a bases de datos y seguridad .

Java fue diseñado para que la ejecución de código a través de la red fuera segura, para lo cual fue necesario deshacerse de herramientas de C tales como los punteros. También se han eliminado aspectos que demostraron ser mejores en la teoría que en la práctica, tales como la herencia múltiple.

La portabilidad fue otra de las claves para el desarrollo de Java, para lograr que las aplicaciones se escriban una sola vez sin la necesidad de modificarlas para que se ejecuten en diferentes plataformas. Esta independencia se alcanza tanto a nivel de código fuente (similar a C++) como a nivel de código binario. La solución adoptada fue compilar el código fuente para generar un código intermedio (*bytecodes*) igual para cualquier plataforma. La JVM (Máquina Virtual de Java), donde reside el intérprete Java, sólo tiene que interpretarlos.

Java aporta elementos que sí resultan interesantes en el desarrollo Web:

a) JDBC (Java DataBase Connectivity): Es un interfaz que permite la comunicación con bases de datos. Consiste de un conjunto de clases e interfaces escritas en Java, que proveen una API (Interfaz de Programación de

Aplicaciones) estándar para desarrolladores de herramientas de base de datos, permitiendo independizar la aplicación de la base de datos que utiliza.

La API JDBC es la interfaz natural a las abstracciones y conceptos básicos de SQL (Lenguaje de Consulta Estructurado): permite crear conexiones, ejecutar sentencias SQL y manipular los resultados obtenidos. Conceptualmente es similar a ODBC (Conectividad de Base de Datos Abierta), pero ésta no es apropiada para usar directamente desde Java porque usa una interfaz en C.

JDBC soporta dos modelos de acceso a base de datos: modelo de dos capas y modelo de tres capas. En el primer caso, la aplicación Java se comunica directamente con la base de datos mediante un controlador JDBC específico para cada SGBD (Sistema Gestor de Bases de Datos) que desee manipular. En el segundo caso, los comandos son enviados a un capa intermedia de servicios, encargado de reenviar las sentencias SQL a la base de datos. Las arquitecturas de dos y tres capas se tratan en el apartado 3.5.3.

Existe un controlador, llamado puente JDBC-ODBC, que implementa las operaciones de JDBC traduciéndolas en operaciones ODBC, con lo cual se facilita acceso a cualquier base de datos cuyo controlador ODBC se encuentre disponible.

b) *Servlets*: Un servlet es una clase Java, embebida dentro del servidor Web, y utilizada para extender la capacidad del servidor. La API de *servlets* provee clases e interfaces para responder a cualquier tipo de requerimiento; en particular, para las aplicaciones que se ejecutan en servidores Web, la API define clases de *servlet* específicas para requerimientos HTTP. No necesitan ser ejecutados como nuevos procesos dado que se ejecutan directamente en el servidor Web. Viven entre sesiones y se puede decir que son persistentes: no es necesario crear un *servlet* por cada requerimiento realizado desde el cliente, sino que corren dentro de éste en múltiples hilos.

Los *servlets* son programas Java que facilitan la funcionalidad de generar dinámicamente contenidos Web. Pueden ejecutarse a través de una línea de comando. A diferencia de los *applets*, no poseen restricciones en cuanto a seguridad. Tienen las propiedades de cualquier aplicación Java y pueden acceder a los archivos del servidor para escribir y leer, cargar clases, cambiar

propiedades del sistema, etc. Del mismo modo que las aplicaciones de programas Java, los *servlets* están restringidos por los permisos del sistema. Son cargados la primera vez que se usan, y permanecen en memoria para satisfacer futuros requerimientos. Tienen un método *init*, donde el programador puede inicializar el estado del *servlet*, y un método *destroy* para liberar los recursos que son mantenidos por el *servlet*.

c) JSP (Páginas de Servidor Java): Proporciona a los desarrolladores un entorno de desarrollo para crear contenidos dinámicos en el servidor usando plantillas HTML y XML, en código Java, encapsulando la lógica que genera el contenido de las páginas. Cuando se ejecuta una página JSP es traducida a una clase de Java, la cual es compilada para obtener un *servlet*. Esta fase de traducción y compilación ocurre solamente cuando el archivo JSP es llamado la primera vez, o después de que ocurran cambios.

JSP y XML tienen un interesante relación; así como pueden generarse páginas HTML dinámicas a partir de una fuente en JSP, pueden generarse dinámicamente de forma análoga documentos XML.

3.5.5.- XML (eXtensible Markup Language).

La familia XML (Lenguaje de Marcas eXtendido) es un conjunto de especificaciones que conforman el estándar que define las características de un mecanismo independiente de plataformas desarrollado para compartir datos. Se puede considerar a XML como un formato de transferencia de datos multiplataforma. Es un subconjunto de SGML y uno de sus objetivos es permitir que SGML genérico pueda servirse, recibirse y procesarse en la Web de la misma manera que actualmente es posible con HTML.

XML ha sido diseñado de tal manera que sea fácil de implementar. No ha nacido sólo para su aplicación en Internet, sino que se propone como lenguaje de bajo nivel (a nivel de aplicación, no de programación) para intercambio de información estructurada entre diferentes plataformas. XML hace uso de etiquetas (únicamente para delimitar datos, no para presentarlos como HTML) y atributos, y deja la interpretación de los datos a la

aplicación que los utiliza. Por esta razón se van formando lenguajes a partir del XML; desde este punto de vista XML es un metalenguaje.

El conjunto de reglas o convenciones que impone la especificación XML permite diseñar formatos de texto para los datos estructurados, haciendo que se almacenen de manera no ambigua, independiente de la plataforma y que en el momento de la recuperación se pueda verificar si la estructura es la correcta.

Para comprobar que los documentos estén bien formados se utiliza un DTD (Definición de Tipo de Documento). Se trata de una definición de los elementos que pueden incluirse en el documento XML, la relación entre ellos, sus atributos, posibles valores, etc. Es una definición de la gramática del documento, es decir, cuando se procesa cualquier información formateada mediante XML, el primer paso es comprobar si está bien formada, y luego, si incluye o referencia a un DTD, comprobar que sigue sus reglas gramaticales.

3.5.6.- XSL (eXtensible Stylesheet Language).

XSL (Lenguaje de hojas de estilo extensible) es una especificación desarrollada dentro del W3C (World Wide Web Consortium) para aplicar formato a los documentos XML de forma estandarizada. Aunque se ha establecido un modo para que puedan usarse hojas de estilo CSS (Hojas de Estilo en Cascada) dentro de documentos XML, es lógico pensar que para aprovechar las características del nuevo lenguaje hace falta tener un estándar paralelo y similar asociado a él.

XSL es un lenguaje para transformar documentos XML, así como un vocabulario XML para especificar semántica de formateo de documentos. Además del aspecto que ya incluía CSS referente a la presentación y estilo de los elementos del documento, añade una pequeña sintaxis de lenguaje de comandos para poder procesar los documentos XML de forma más cómoda. XSL permite añadir lógica de procesamiento a la hoja de estilo.

La idea es asociar al documento XML con una hoja de estilo y a partir de esto visualizar el documento XML en cualquier plataforma: *PalmPC*, *PC*, etc. con *Internet Explorer*, *Netscape*, etc. y con el aspecto (colores, fuentes, etc) que se quiera utilizar.

En esencia, XSL son dos lenguajes: uno de transformación y otro de formateo. El lenguaje de transformación permite transformar un documento XML en otro con diferente formato, como HTML o texto plano, o bien en otro documento XML. El lenguaje de formateo no es más que un vocabulario XML para especificar objetos de formateo.

Al igual que con HTML, se pueden especificar las hojas de estilo, CSS o XSL, dentro del propio documento XML o haciendo referencia a ellas en forma externa. Esto es muy útil para mover datos de una representación XML a otra representación, basada en correo electrónico, intercambio de datos electrónicos, intercambio de metadatos, y alguna aplicación que necesite convertir datos entre diferentes representaciones de XML a otro tipo de representación.

La gran ventaja de utilizar XML y XSL es que los datos y la presentación de estos quedan en dos archivos diferentes**.**

3.6.- SISTEMAS GESTORES DE CONTENIDOS: ALTERNATIVAS.

La aparición de los gestores de contenidos para la gestión y administración de portales ha sido una verdadera revolución en Internet. Desde sus comienzos, la introducción de información en la red dependía de la disponibilidad de un técnico, o bien pasaba por la formación en HTML del personal implicado. Este hecho llevaba, en múltiples ocasiones, a que la información publicada estuviera temporal o totalmente obsoleta, debido al cuello de botella que se formaba en estas personas cualificadas para introducir contenidos en el sitio Web.

Los gestores de contenidos abren a múltiples usuarios con conocimientos básicos de informática la posibilidad de publicar información. De esta forma, el número de publicadores de contenidos se multiplica, favoreciendo el dinamismo de la documentación publicada así como la introducción de volúmenes mayores de información.

El mercado no ha sido ajeno a las posibilidades que se ofrecen, y se ha lanzado a ofrecer productos de portales, de diversa índole, pero con el común denominador de la existencia de un gestor de contenidos (CMS, Content Management System). Así han aparecido Broadvision, Vignette, Microsoft CMS, Plone, Typo3, etc.

Un sistema gestor de contenidos es una herramienta que permite a las empresas y entidades la creación, desarrollo y mantenimiento de portales Internet e Intranet, permitiendo la introducción de contenidos de manera fácil e intuitiva, separando para ello la labor del personal técnico del trabajo a realizar por los publicadores de información.

En [BOICO 02] se apunta que el propósito de un CMS es obtener contenidos de una manera sencilla, sin esfuerzos, sabiendo qué contenidos son los que se tienen y así poder difundirlos a una gran variedad de lugares automáticamente. El concepto de contenido está definido como "*la información en bruto se convierte en contenido cuando se ofrece de una forma usable y con un objetivo definido*".

Un CMS es un sistema complejo en el que interactúan varias partes. La información en bruto fluye a través del sistema de colección y se transforma en contenidos. El sistema de gestión la almacena y el de publicación la convierte en publicaciones. La Figura 3.11 ilustra el esquema general de un CMS .

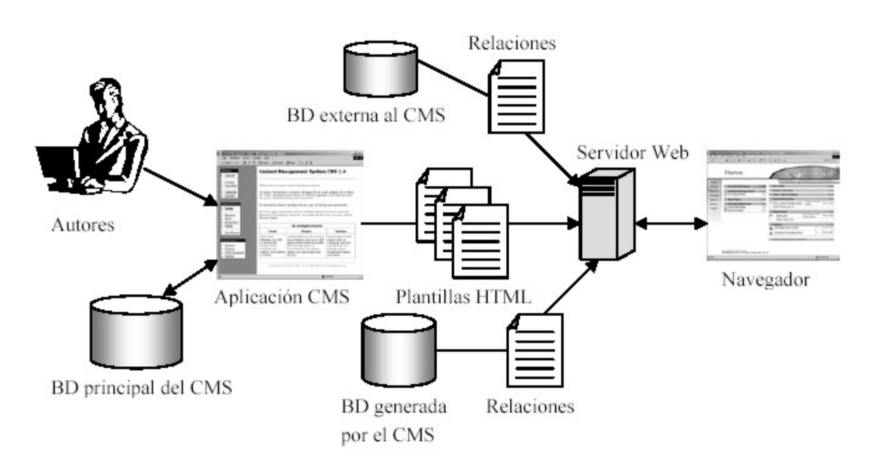


Figura 3.11. Esquema general de un CMS.

Las principales funcionalidades que ofrecen este tipo de herramientas son:

- Ofrecer a los proveedores de contenidos la posibilidad de crear, gestionar y publicar contenidos, permitiendo a los departamentos técnicos el desarrollo rápido de un portal escalable y dinámico.
- Compartir el mismo contenido para ser visualizado desde diferentes dispositivos, así como permitir la existencia de diferentes contenidos mostrados sobre la misma interfaz. Es decir, posibilidades multidispositivo y multilenguaje.
- Realizar un rápido desarrollo y puesta en marcha de un portal, con herramientas más potentes que las utilizadas tradicionalmente.

Los siguientes apartados presentan cuatro ejemplos de sistemas gestores de contenidos. CMS, como ejemplo de producto comercial de Microsoft; Typo3, como ejemplo de producto de código abierto con licencia GPL y desarrollado en el popular PHP. Zope, con licencia ZPL de software libre y código abierto y desarrollado en Python y Plone, un gestor de contenidos de reciente creación basado en Zope con licencia GPL. Los dos primeros se presentan brevemente, incluyendo una visión general y sus características más importantes. Zope se describe con mayor extensión y detalle por tratarse del sistema gestor de contenidos elegido para desarrollar el Generador Automático de un Sistema de Publicación Web para Institutos de Enseñanza Secundaria (GASPWIES).

3.6.1.- CMS (Content Management Software).

CMS es un sistema gestor de contenidos que permite que las empresas, compañías y entidades desarrollen, implementen y administren sitios Web de gran envergadura, de una manera rápida y eficiente. Al simplificar los procesos de publicación Web, se puede reducir el coste del mantenimiento los de sitios, al transferir a los propios usuarios la posibilidad de la gestión del contenido.

CMS es un producto de Microsoft. Está desarrollado con la tecnología .NET. Tiene la posibilidad de integrarse con Microsoft Visual Studio® .NET y Microsoft .NET Enterprise Servers, Microsoft Commerce Server y Microsoft SharePoint™ Portal Server, lo que permite desarrollar una solución de administración de contenido Web de una manera rápida y rentable.

3.6.1.1.- *Características.*

CMS ofrece una variedad de características para la aportación y entrega de contenido, desarrollo de sitios y administración de sitios empresariales para permitir que los negocios creen, implementen y administren efectivamente sitios Web de Internet, Intranet y Extranet. Las más importantes son:

a) *Creación, publicación y almacenamiento de contenido.* Proporciona herramientas sencillas que permiten crear y publicar contenido personalizado y de gran calidad, directamente en sitios Web. El modelo de publicación distribuido incorpora una aceptación de flujo de trabajo con múltiples niveles, programación y archivo automático de contenido y clasificación de contenido. Los desarrolladores pueden crear plantillas de páginas administradas centralizadamente y procesos de publicación que aseguren la consistencia a través del sitio. Almacena todo el contenido en XML, HTML y objetos de contenido binario para una tener una máxima flexibilidad. Los objetos se almacenan en un repositorio de Microsoft SQL Server. Dispone de dos herramientas principales de administración de contenidos Web:
   - *Web Author Client*: permite crear, editar y publicar contenido directamente dentro de un explorador de Internet. Cuando los usuarios necesitan agregar o editar contenido Web, pueden navegar a la ubicación en el sitio. Una vez que han sido autentificados como creadores de contenido en la herramienta de flujo de trabajo, pueden editar el contenido, o bien pueden seleccionar una plantilla de páginas desarrolladas previamente para crear nuevo contenido. Las páginas Web pueden verse previamente antes de que se envíen a revisión.
   - *Authoring Connector for Microsoft Office:* permite crear y publicar contenido directamente desde Microsoft Word, al usar Authoring Connector for Microsoft Office.

b) *Desarrollo e implementación de sitios Web*: Se integra con Visual Studio .NET. El entorno de desarrollo incluye: proyectos, plantillas, controles de servidor, integración dinámica del código fuente a través de objetos de plantillas, etc.

c) *Escalabilidad y fiabilidad*: Ha sido probado para dar servicio a más de 100 millones de páginas dinámicas por día [ENGLISH 03]. Es escalar verticalmente (aumentando el número de procesadores en el servidor) para aumentar el rendimiento en situaciones de alta demanda de tráfico de sitios Web. Se puede escalar horizontalmente incorporando servidores adicionales en entorno de balanceo de carga, pudiendo responder así a las demandas más elevadas de servicio y garantizando el mayor tiempo de operación, al minimizar el impacto de cualquier condición de fallo de hardware.

d) *Otras características:*

- Posibilita un correcto proceso de revisión y aprobación mediante un sistema de flujo de trabajo *(workflow)* parametrizable que incluye numerosas funciones de verificación y control.
- Permite que los usuarios realicen auditorias con fines específicos al almacenar automáticamente revisiones de contenido y versiones de páginas.
- Permite construir páginas dinámicamente para personalizar el contenido en base al perfil del usuario, análisis a través de clics, preferencia de idioma, dispositivos y exploradores de Internet.
- Intercambia las plantillas rápidamente para presentar contenido en diferentes estilos y diseños para soportar diversos grupos de dispositivos de exploración.
- Puede emplear modelos existentes de autentificación o integrar modelos personalizados al usar extensiones de *ASP.NET authentication*.
- Soporta hardware y software de equilibrio de carga de red.
- Administra contenido en formato XML.
- Permite administrar los estilos de presentación mediante hojas de estilo XSLT.
- Soporta la presentación de contenido y la funcionalidad dentro del sistema como servicios Web XML estándar.
- Soporte total para servicios de Microsoft Active Directory.

3.6.2.- Typo3

TYPO3 es un CMS de código abierto y desarrollado bajo licencia GPL utilizando el lenguaje PHP y el sistema gestor de bases de datos relacionales MySQL. Se trata de un sistema modular, flexible, potente y que posee una curva de aprendizaje muy suave para los usuarios, lo que lo capacita como una plataforma de desarrollo de cualquier tipo de solución Web. La lista de prestaciones es muy extensa, y está continuamente en crecimiento debido a la gran comunidad de desarrolladores que contribuyen al producto.

En [ICTI] se enumeran las ventajas de la utilización de Typo3:

- Eficacia en la gestión de contenidos y creación de flujos de trabajo simplificados.
- Flexibilidad debido a que se adapta a diferentes necesidades e instalaciones.
- Rapidez y simplicidad gracias a la utilización de una interfaz práctica e intuitiva.
- Manipulación de imágenes para crear menús más dinámicos.
- Multilingüe, ya que tiene total integración en más de 16 lenguas.
- Extrema facilidad de manejo.
- Calidad profesional de los resultados.

3.6.2.1.- *Características.*

Typo3 está reconocido como uno de los mejores sistemas gestores de contenidos de código abierto. Es gratuito al tener licencia GPL y lleva varios años funcionando por lo que resulta totalmente estable. De hecho, existen numerosos portales desarrollados con Typo3 [TYPO3]. Está en constante crecimiento por la participación de una amplia comunidad de desarrolladores distribuidos en todo el mundo y dispone de documentación para distintos niveles de utilización en [TYPO3]. Otras características técnicas reseñables son las siguientes:

a) *De interés para usuarios:*

- Es posible publicar desde cualquier sitio independientemente de la plataforma utilizada solamente con un navegador.

- Un editor de texto enriquecido permite a los usuarios formatear el texto, insertar imágenes y crear enlaces internos y externos exactamente igual que en Microsoft Word o cualquier otro procesador de textos habitual.
- Ofrece gestión automática de imágenes, asegurándose de que tiene el tamaño y la calidad correctos para el sitio Web. La tecnología "*Magic Image*" en el asegura un control total sobre la calidad, las proporciones y el tamaño.
- La separación entre campos principales y secundarios en los formularios permite mostrar al usuario únicamente las opciones más frecuentes, mientras que se ocultan las opciones avanzadas para usarlas sólo cuando se requiera expresamente.
- El conocido concepto de cortar y pegar se puede utilizar mediante el portapapeles interno para manejar reubicaciones y repeticiones de todos los objetos internos.
- Los menús contextuales dan al usuario acceso directo a las funciones más utilizadas de cada objeto con un simple clic. Existen asistentes para la creación de tablas, formularios de correo, boletines, etc.
- Todas las categorías de tipo de contenido se añaden por medio de formularios fáciles de comprender.
- Las páginas se ordenan jerárquicamente de manera clara y bien conocida por cualquier usuario de PC, siguiendo el formato del sistema local de archivos de carpetas y ficheros.
- La interfaz del panel de control ofrece ayuda sensible al contexto desde pequeños iconos.

b) *De interés para administradores:*

- Existen perfiles predeterminados listos para trabajar combinados con un nivel extremo de opciones de personalización (definidas bajo sintaxis *TypoScript*) que proporcionan la posibilidad de un ajuste muy fino del panel de control y del entorno de trabajo al que tiene acceso cada usuario y cada grupo o subgrupo.
- El control de acceso a las páginas se puede configurar en función de usuario, propietario o grupo (de manera parecida a UNIX).

- Permite autorizar a los usuarios y grupos a acceder y editar únicamente ciertos objetos y módulos de ciertas páginas en ciertas secciones del sitio Web. Se pueden asignar perfiles sencillos o muy complejos.
- La herramienta de supervisión simplifica enormemente el análisis de los permisos de un gran número de usuarios.
- Tiene un amplio abanico de estadísticas, desde simples contadores internos de visitas hasta análisis avanzado de ficheros de registro exportados.
- Actúa como un gestor avanzado de base de datos, con un panel que proporciona total control para manejar el contenido. Otra aplicación PHP muy popular, el gestor de bases de datos phpMyAdmin, se ha integrado y preconfigurado con la base de datos utilizada.

c) *De interés para desarrolladores:*

- Cuenta con una avanzada tecnología de plantillas, basada en su propio lenguaje de configuración, *TypoScript*, cuyo análisis por el sistema construye las páginas implementadas, incluyendo menús gráficos, cabeceras y más. *TypoScript* se ordena en plantillas en el editor integrado, y proporciona la posibilidad de controlar y diseñar el sitio al completo desde todos los niveles desde una perspectiva abstracta, en lugar de una configuración a nivel de página.
- Las plantillas pueden organizarse jerárquicamente, y permiten ignorar algunos parámetros para tenerlos en cuenta en niveles posteriores. Otro tipo de plantillas son las de archivos HTML para configurar rápidamente la presentación de noticias, tablón de anuncios, etc.
- Si se necesita código personalizado, el asistente para principiantes ayuda a añadir funcionalidades y guía al desarrollador para seguir las líneas maestras para la creación de módulos. Esto asegura la adhesión a las reglas de arquitectura de Typo3, lo que supone una ventaja en cuanto que facilita actualizaciones separando el núcleo y las extensiones.
- Es posible tener cualquier conexión imaginable con fuentes de datos externas mediante la integración de *scripts* PHP propios.
- Se pueden integrar elementos como vídeo, audio, animaciones flash, applets java, etc. También se puede incorporar de forma sencilla código HTML personalizado.

- Se pueden crear Intranets, Extranets y secciones protegidas por contraseña en el sitio Web.
- El motor de plantillas permite la detección del navegador, versión imprimible, versiones de PDAs, etc., uso de hojas de estilo CSS, WML para teléfonos móviles, XML para intercambio de datos, SGML para impresión... Cualquier formato puede ser generado por las extensiones de PHP.

d) *Tecnología*: La interacción con la aplicación se realiza con un navegador. Por tanto, no requiere ningún software especial en el lado del usuario. Cualquier navegador actual con soporte de gráficos es válido. Prestaciones como el procesamiento de imágenes requiere de software adicional, pero generalmente están disponibles disponible en el servidor como *Gdlib, Freetype* e *ImageMagic*. El contenido se almacena en una base de datos relacional muy rápida como es MySQL.

e) *Seguridad.* Pueden configurarse descargas seguras de archivos. El envío y almacenamiento de contraseñas se encripta con *md5*. Para Intranets, Extranets y áreas de administración puede aplicarse filtrado IP. Para la transmisión de datos encriptados se puede usar SSL.

f) *Plataforma*. Typo3 funciona óptimamente bajo el servidor Apache, usando el motor de *scripts* PHP 4 y MySQL como motor de base de datos. Todos estos componentes están disponibles en varias plataformas (Linux, Windows, FreeBSD, etc..) y, por consiguiente, lo hacen un CMS realmente multiplataforma. No obstante, la plataforma de funcionamiento más común por su robustez y rendimiento es Linux.

3.6.3.- Zope.

Zope forma parte de una nueva generación de servidores de aplicaciones que facilitan el desarrollo de sitios Web. Funciona sobre todas las plataformas UNIX-LINUX así como en Windows NT, 2000, 2003 y XP. Está escrito en Python, un potente lenguaje orientado a objetos. Es el producto estrella de la compañía *Zope Corporation*.

A diferencia de las tecnologías basadas en páginas Web dinámicas como ASP o PHP, Zope es una plataforma completamente orientada a objetos. Facilita la separación de las capas de presentación, control y almacenamiento de datos, incluye un amplio conjunto de objetos reutilizables y lleva integrado un modelo de seguridad robusto.

Sus creadores, *Digital Creations* y un gran número de usuarios que forman la comunidad Zope, lo definen como "*una plataforma de alto rendimiento para desarrollar aplicaciones Web dinámicas*". Sus características más importantes son:

a) Puede integrarse con buena parte de los servidores Web (Apache, Internet Information Server, etc) y como alternativa incluye un servidor Web propio (Medusa).
b) Su arquitectura se basa en mecanismos de intercambio de datos con el servidor de datos Zope.
c) Un interfaz Web para el desarrollo de aplicaciones (ZMI).
d) Una base de datos orientada a objetos (ZODB), en la cual se guardarán todos los desarrollos/datos/transacciones que se realicen en Zope.
e) Soporte de estándares abiertos: SQL, ODBC, XML, FTP, HTTP, etc.
f) Programación dinámica con sus lenguajes de *script* (*Zope Page Template* - ZPT, *Document Template Markup Language* - DTML) o a través de lenguajes de programación como Python o Perl.
g) API sencilla de acceso a la base de datos.

La Figura 3.12 [MIGUEL 01] muestra la arquitectura de Zope.

De todas sus características destaca la orientación a objetos. Una URL a un recurso Web es realmente un camino a un objeto que forma parte de una jerarquía. El protocolo HTTP permite enviar mensajes al objeto y obtener su respuesta. En Zope todos son objetos: carpetas, documentos, imágenes o sentencias SQL. Un ejemplo: la dirección http://localhost:8080/demo/index_html apunta a un objeto llamado `index_html` situado en una carpeta llamada `demo`.

En este proyecto se ha elegido Zope como herramienta de desarrollo del Generador Automático de un Sistema de Publicación Web para Institutos de Enseñanza Secundaria

(GASPWIES). En los siguientes subapartados se realiza una introducción a los aspectos más importantes del sistema Zope.

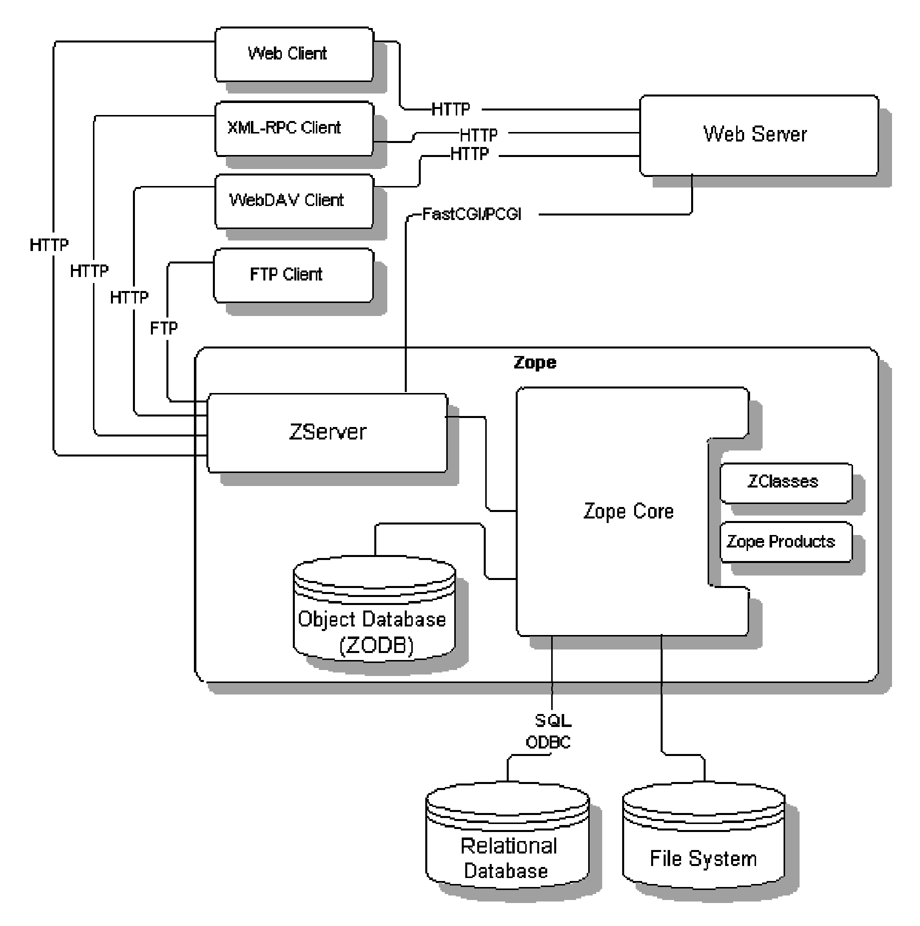


Figura 3.12. Arquitectura de Zope.

3.6.3.1.- *Puesta en funcionamiento.*

Zope es una aplicación de software libre que está disponible en la dirección http://www.zope.org/Products. Según la plataforma sobre la que se instale el método es diferente. En Windows basta con ejecutar el archivo descargado (por ejemplo, `Zope2.7.0-win32.exe`) y en Linux se debe descomprimir el archivo descargado y ejecutar el *script* de instalación (`install`).

Para ponerlo en funcionamiento debe ejecutarse el *script* correspondiente (`start` en Linux o `start.bat` en Windows, aunque normalmente en Windows existirá un icono

de acceso directo. El resultado de la ejecución se muestra en la Figura 3.13 (puede variar en función de la versión instalada, en este caso la 2.7.0).

```
Zope 2.7.0
C:\Documents and Settings\Juan Carlos\Escritorio>"C:\Archivos de programa\Zope-2
.7.0\bin\python.exe" "C:\Archivos de programa\Zope-2.7.0\lib\python\Zope\Startup
\run.py" -C "C:\Zope-Instance\etc\zope.conf"
------
2004-08-28T12:08:31 INFO(0) ZServer HTTP server started at Sat Aug 28 12:08:31 2
004
        Hostname: localhost
        Port: 8080
------
2004-08-28T12:08:31 INFO(0) ZServer FTP server started at Sat Aug 28 12:08:31 20
04
        Hostname: jcpp
        Port: 8021
------
2004-08-28T12:08:37 DEBUG(-200) FileStorage create storage C:\Zope-Instance/var/
Data.fs
------
2004-08-28T12:08:41 DEBUG(-200) TemporaryStorage create storage temporary storag
e for sessioning
------
2004-08-28T12:08:41 BLATHER(-100) ZODB Commiting subtransaction of size 5386
------
2004-08-28T12:08:42 INFO(0) Zope Ready to handle requests
```

Figura 3.13. Puesta en marcha del servidor Zope.

Para entrar a la aplicación Zope debe abrirse una instancia de un navegador de Internet y teclear la URL http://localhost:8080/manage. El interfaz de Zope está escrito íntegramente en HTML, por lo que tanto Mozila como Netscape Navigator y Microsoft Internet Explorer son válidos para interactuar con Zope. Es necesario validarse con el usuario y la contraseña de administración facilitados en el proceso de instalación.

El aspecto del interfaz de administración de Zope (ZMI) presenta el aspecto que puede verse en la Figura 3.14.

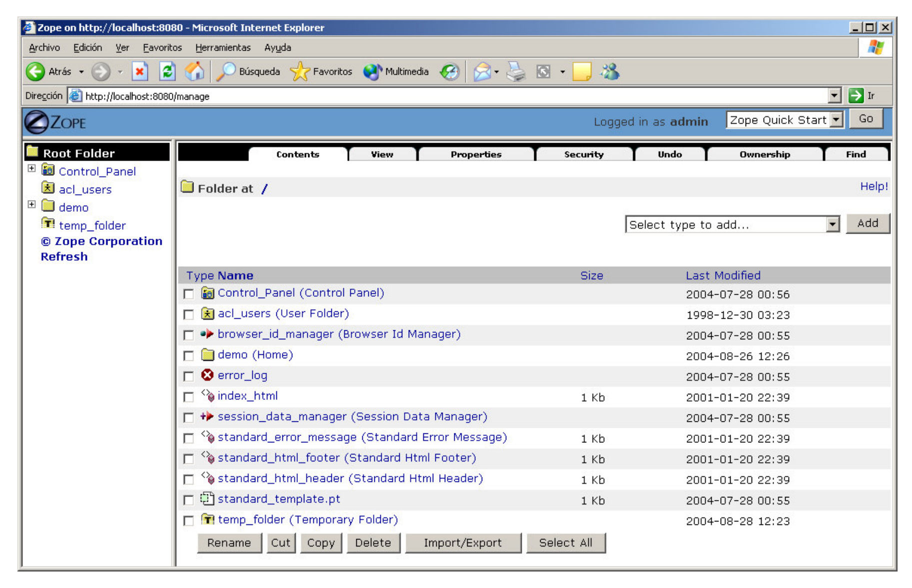


Figura 3.14. Interfaz de administración Zope (ZMI).

El interfaz es totalmente intuitivo de usar si se está acostumbrado a la navegación en Internet. Está distribuido en tres marcos: el superior permite desconectarse del sistema, establecer las preferencias de visualización y acceder a una vista rápida del sistema; el izquierdo permite la navegación por los objetos Zope y el derecho visualiza las propiedades del objeto elegido en la navegación, así como, permite realizar operaciones sobre él.

El objeto *Panel de Control* permite la administración y monitorización del servidor Zope. Ver Figura 3.15.

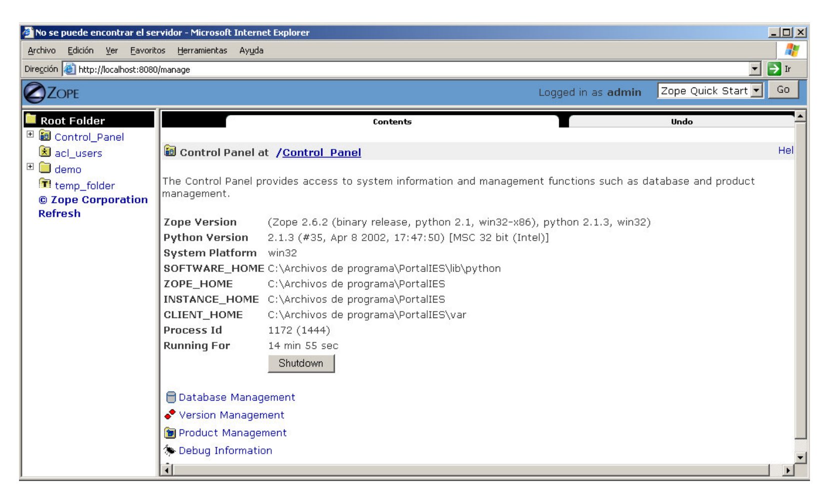


Figura 3.15. Panel de Control de Zope.

El ZMI ofrece una completa ayuda en línea (en el enlace *Help*), así como un tutorial para familiarizarse con el sistema (instalable siguiendo las instrucciones del *Zope Quick Start*).

3.6.3.2.- *Objetos básicos.*

En general, los objetos Zope tienen tres tipos de rol:

- *Contenido:* tales como documentos, imágenes y diferentes tipos de datos binarios y de texto.
- *Lógica:* permiten realizar funciones de *scripting* utilizando Python, Perl y SQL.

- *Presentación:* permiten controlar la vista del sitio Web creado. Se incluyen dos tipos de objeto que permiten utilizar el lenguaje propio de Zope DTML (*DTML Documents* y *DTML Method)* y las plantillas Zope (ZPT).

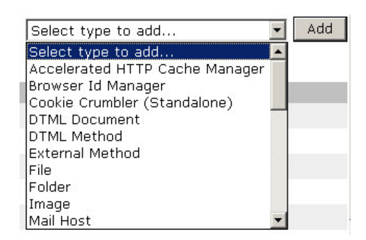


Figura 3.16. Añadir un objeto Zope.

Para añadir un nuevo objeto a un contenedor (carpeta) se debe elegir de la lista desplegable que ofrece Zope y pulsando el botón `Add`. A continuación se expone la funcionalidad de los objetos básicos de Zope:

a) *Carpeta (Folder)*: su propósito es contener otros objetos, con el objetivo de organizarlos y agruparlos. Pueden contener toda clase de objetos, incluyendo otras carpetas que pueden formar una estructura arbórea. Se utilizan para crear la estructura del sitio Web que se construirá. La estructura es fundamental en el éxito o fracaso del sitio. Las carpetas pueden exportarse para importarlas en otras carpetas del mismo o de otros servidores Zope.

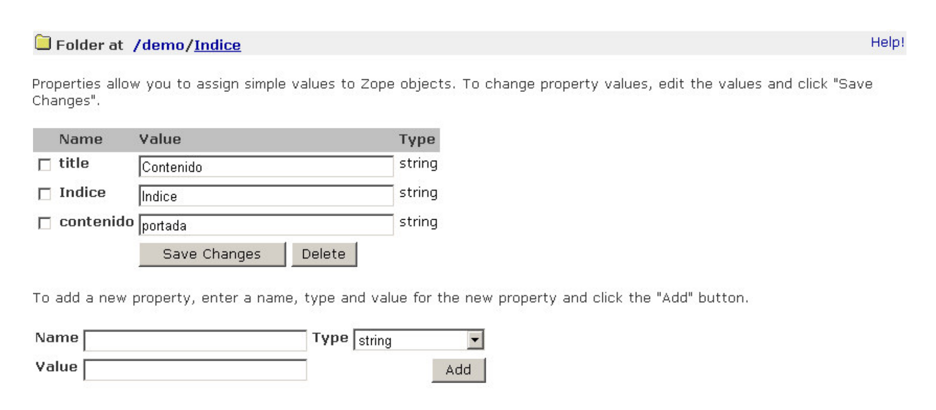


Figura 3.17. Propiedades de los objetos Zope.

Las carpetas, como todos los objetos Zope tienen propiedades. En la Figura 3.17 pueden verse las propiedades de la carpeta `/demo/Indice`. La propiedad `title` es heredada de la clase progenitora de todos los objetos básicos

(`ObjectManager`) y las propiedades `indice` y `contenido` son propias de la carpeta. Como puede verse, con el ZMI pueden añadirse nuevas propiedades y modificarse y eliminarse existentes.

b) *Archivo (File):* almacenan datos como los ficheros del sistema de archivos del sistema operativo. Software, video, audio, documentos, hojas de cálculo, ficheros Flash, applets, etc. son contenidos que pueden almacenarse en objetos de este tipo. Cada archivo tiene una propiedad llamada `content_type` que contiene el tipo MIME del archivo. El contenido del archivo puede "subirse" de un fichero almacenado en cualquier soporte.



Figura 3.18. Subir contenido para un objeto Zope.

c) *Imagen (Image):* almacenan gráficos de tipo GIF, JPEG, BMP y PNG. Son objetos similares a los archivos, pero añaden las propiedades `height` (altura) y `width` (anchura) del tamaño de la imagen. También es posible "subir" el contenido del archivo con el botón `Upload`.

d) *Métodos*: son objetos que tienen un contenido ejecutable especial. También pueden nombrarse como *scripts*. Existen dos clases de métodos: *DTML Methods* (usados para definir plantillas de presentación) y *SQL Methods* (usados para contener cláusulas SQL que se aplicarán al sistema gestor de base de datos sobre el que esté montada la capa de almacenamiento). Los objetos básicos como carpetas, archivos e imágenes tienen métodos asociados según su árbol de herencia. Por ejemplo, las carpetas tienen un método `objectValues` que retorna una lista de los objetos que contiene la carpeta. Un sencillo ejemplo de método DTML que muestra el identificador y el título de los objetos de la carpeta sobre la que se ejecuta el método se escribiría:

```
<ul>
 <dtml-in objectValues>
  <li><dtml-var getId>,<dtml-var title></li>
 </dtml-in>
</ul>
```

e) *Documentos DTML (DMTL Document)*: son páginas Web que pueden contener comandos DTML. La combinación de HTML con DTML genera páginas Web dinámicas. Un sencillo ejemplo que muestra un pié de página:

```
<i>Gracias por su visita, © <dtml-var portal></i>
```

Suponiendo que el contenido anterior está almacenado en un objeto llamado `piepag` y situado en la raíz del servidor, la página puede visualizarse directamente con la URL http://localhost:8080/piepag. También puede invocarse a un documento DTML desde un método DTML o desde otro documento DTML. Por ejemplo,

```
<html>
<title>Ejemplo</title>
<body>
  <p>Bienvenido a nuestro portal,
    <dtml-var AUTHENTICATED_USER>
  </p>
<dtml-var piepag>
</body>
</html>
```

f) *Plantillas Zope (Zope Page Template):* este objeto se incluye a partir de la versión 2.5 de Zope. Permite definir presentación dinámica para una página Web. El HTML de la página se convierte en dinámico incluyendo etiquetas especiales XML. Su utilización separa la lógica de la presentación e intencionadamente no permiten usarlas como un lenguaje de propósito general. Un ejemplo de ZPT:

```
<html>
<body>
<p>
 El título de la plantilla es:
 <b tal:content="template/title">titulo</b>.
</p>
</body>
</html>
```

g) *Script Python*: son objetos que contienen código Python. Un ejemplo que retorna la fecha del sistema sería:

```
from DateTime import DateTime
fecha='%s / %s / %s' %
 (DateTime().day(),DateTime().mm(),DateTime().year())
return fecha
```

3.6.3.3.- Contenido dinámico.

Zope facilita tres métodos de implementar las páginas Web con contenido dinámico: su lenguaje propio de *scripting* llamado DTML y utilizable en sus tipos de objeto *DTML Method* y *DTML Document,* sus plantillas ZPT y los *scripts* Python.

Este apartado pretende realizar una introducción a las características más importantes de DTML y ZPT.

3.6.3.3.1.- *DTML (Document Template Markup Language)* .

DTML es un lenguaje del lado servidor. Es el lenguaje de *script* y presentación basado en etiquetas de Zope. DTML genera, controla y da formato al contenido de manera dinámica. No es ortodoxo para seguir el patrón de tres capas ya que no separa la presentación de la lógica. Por tanto, se utilizará cuando los resultados a obtener sean simples.

DTML está pensado para ser usado por gente familiarizada con HTML y *scripting* básico de Web. No es adecuado para programación compleja. Permite reutilizar contenido genérico (cabeceras, pies, etc.) La posibilidad de dar formato a cualquier tipo de datos textuales hace que DTML sea una herramienta de presentación muy potente, ya que permite modificar la lógica sin tener que cambiar la presentación. En [LATTEIER 00] se plantean dos dudas que conviene aclarar.

¿Cuándo no conviene utilizar DTML? No se trata de un lenguaje de programación de propósito general (por ejemplo, no permite crear variables de manera sencilla). No facilita la creación de algoritmos (para eso, mejor utilizar *scripts* en Python o en Perl). Tampoco es una buena opción a la hora de procesar cadenas de texto (mejor Perl y Python). No es buena idea utilizarlo para plantillas genéricas (para eso es mejor utilizar ZPT).

¿Cuándo debe utilizarse un método DTML y cuándo un documento DTML? Si es contenido, se debe usar documento. Si ha sido pensado para ser mostrado por otros objetos, también un documento. Si el objeto requiere propiedades propias, también un documento porque los métodos no tienen propiedades. Si está pensado para presentar otros objetos, es

mejor un método. Si el objeto necesita transparencia con respecto a su contexto, también es mejor un método, ya que estos objetos actúan como si fueran adjuntados al objeto que les llama o que les contiene.

La sintaxis de DTML es similar a la de HTML; es un lenguaje de marcado de etiquetas. Se puede mezclar con otros lenguajes de marcado, en general HTML, pero se pueden generar otros tipos de texto (XML, mensajes de correo o información textual, etc.) DTML contiene dos tipos de etiquetas: *singleton* (no se cierran, como por ejemplo `dtml-var`) y *de bloque* (se abre y se cierra un bloque mediante una etiqueta, como por ejemplo `dtml-in`).

Todas las etiquetas DTML tienen atributos. Estos facilitan información de cómo ha de funcionar la etiqueta. Algunos atributos son opcionales y otros no tienen valor. Algunas etiquetas de DTML son:

- `var`: introduce variables en métodos y documentos DTML.
- atributo `expr` en etiqueta `var`: permiten llamar a métodos Python desde DTML.
- `if / else / elif / unless`: evalúan condiciones y actúan en consecuencia.
- `in`: itera sobre una secuencia de objetos.
- `in / with / let`: modifican la prioridad de los espacio de nombres.
- c`all`: llama a métodos sin devolver valor de retorno.
- `comment`: documenta código DTML con comentarios DTML.
- `tree`: genera árboles dinámicos en HTML.
- `sendmail / mime`: envío de correos electrónicos.
- `raise / try / finally / except`: gestión de excepciones.

3.6.3.3.2.- *ZPT (Zope Page Template).*

ZPT es un objeto que permite definir presentación dinámica para una página Web. También es un lenguaje de s*cript* del lado servidor. Zope ejecuta las etiquetas propias de ZPT y el resultado (HTML) lo envía al navegador cliente. A diferencia de DTML, ZPT se utiliza solamente para la presentación, por lo que resulta ideal para ser utilizado con el patrón de tres capas.

ZPT permite a los desarrolladores una gran flexibilidad, separando la presentación de la lógica de manera que resulta sencillo alterar una sin necesidad de modificar la otra. Es un generador dinámico de HTML/XML. Su objetivo principal es facilitar el trabajo a los diseñadores y a los programadores. Si Zope ya tiene DTML, ¿porqué ZPT?. ZPT se incluye en la versión 2.5 para "suplir" algunas quejas. Por ejemplo, DTML no permite la escalabilidad por no separar la lógica de la presentación. Además, una página con el DTML insertado, no es una página HTML válida. En cambio, una ZPT es una página HTML válida.

ZPT está formado por tres componentes:

a) *TAL (Template Attribute Language)*. Es el lenguaje de plantillas que emplea atributos especiales en las etiquetas HTML para realizar distintas acciones como operaciones con variables, repetición de etiquetas y sustitución de contenidos. Las sentencias de TAL son atributos XML que pueden aplicarse a documentos XML o HTML. Las principales sentencias TAL son:
   - `tal:attributes`: cambian dinámicamente atributos.
   - `tal:define`: permiten definir variables.
   - `tal:condition`: evalúa el cumplimiento de una condición lógica.
   - `tal:content`: sustituye el contenido de un elemento.
   - `tal:omit-tab`: permite eliminar un elemento, de forma que no se incluya su contenido.
   - `tal:on-error`: maneja la ocurrencia de errores.
   - `tal:repeat`: repite elementos.
   - `tal:replace`: sustituye el contenido de un elemento y elimina el contenido del elemento.

   Se debe destacar que cuando en una misma etiqueta HTML se incluyen varias TAL, existe un orden predefinido de ejecución: `define, condition, repeat, content` ó `replace, attributes` y `omit-tag`.

b) *TALES (Tal Expression Syntax)*. Describe qué expresiones se pueden utilizar para proporcionar datos a TAL y a METAL. Las más importantes son:

- `path`: localizan un valor por su path.
- `exists`: evalúa si un path es válido.
- `nocall`: localizan un objeto por su path.
- `not`: niegan una expresión.
- `string`: dan formato a una cadena.
- `python`: ejecuta una expresión Python.

c) *METAL (Macro Expansion TAL)*. Lenguaje de atributos para el preprocesado estructurado de macros. Las sentencias más importantes son:

- `metal:define-macro`: define una macro, por ejemplo para incluir trozos de página, que se podrá utilizar en adelante (para encabezados, pies de página, etc.).
- `metal:use-macro`: usa una macro.
- `metal:define-slot`: permite incluir una macro dentro de otra.
- `metal:fill-slot`: un *slot* es una posición que se puede rellenar dinámicamente cada vez que se utilice la macro.

En el anexo I pueden consultarse ejemplos de uso de ZPT y DTML extraídos de la aplicación GASPWIES.

3.6.3.4.- Adquisición.

En Zope se utiliza el término adquisición (*acquisition*) para referirse al comportamiento dinámico de la búsqueda de métodos y propiedades de los objetos. Puede considerarse un término similar a la herencia en orientación a objetos, aunque no es lo mismo.

Contenido de la carpeta Ejemplo
dos(Carpeta dos)
⊟ uno(Carpeta uno)
unodos(Carpeta unodos)

Figura 3.19. Ejemplo de adquisición.

Cuando se hace referencia a un objeto, propiedad o método, Zope lo intenta localizar en el contexto del objeto referenciador. Si lo encuentra, lo retorna, lo ejecuta, etc. en función del tipo de objeto, pero si no lo encuentra, lo intenta localizar buscando en los contenedores progenitores del objeto referenciador. Un ejemplo puede ayudar a la comprensión de este concepto.

Caso 1.- Los objetos carpeta *(Folder*) descienden de la clase `ObjectManager` que posee un método llamado `objectValues`. Si un objeto de la carpeta `dos` hace una llamada al método, el método retorna los objetos que contiene esta carpeta. Sin embargo, si se llama desde un objeto de la carpeta `unodos` retornará una lista de objetos con el contenido de `unodos`.

Caso 2.- Si la carpeta `unodos` posee una propiedad llamada `color` y un objeto situado en `unodos` hace referencia a la propiedad `color`, Zope retornará el valor de esta propiedad. Si la carpeta `unodos` no tuviera la propiedad `color`, Zope la intentaría encontrar en la carpeta `uno`, en la carpeta `ejemplo` y sus antecesoras si las hubiera (y en este orden), hasta encontrarla o visualizar un mensaje de error.

Caso 3.- Si la carpeta `dos` tiene una propiedad llamada `tamano` y un objeto situado en la carpeta `unodos` hiciera una petición de la propiedad `dos/tamano`, Zope intentaría encontrar, primeramente, un objeto llamado `dos`. Como en el contexto llamador no lo encuentra, asciende hasta llegar al contenido de la carpeta `ejemplo` donde existe la carpeta `dos` y a continuación comprueba si esta carpeta tiene la propiedad `tamano` para retornarla o visualizar el correspondiente mensaje de error.

La adquisición es un patrón muy potente de desarrollo Web. Si en varias secciones diferentes de la estructura de una Web se necesita visualizar un encabezado diferente, por ejemplo, no es necesario tener dos páginas diferentes. La solución correcta es tener una página que invoca a un método que visualice el encabezado. Existirán tantos métodos como secciones, de forma que dependiendo de donde se invoque a la página de presentación el método que retorne el encabezado será uno u otro.

3.6.3.5.- *Seguridad y usuarios.*

Las aplicaciones Web necesitan seguridad. Es necesario controlar quién accede a la aplicación y saber qué hacen. La seguridad es un elemento imprescindible en el diseño de las aplicaciones Web. Debe proporcionar un mecanismo de privacidad de la información y de protección contra acciones malintencionadas que vulneren la integridad de la aplicación. Zope proporciona políticas de seguridad en todos los aspectos de la construcción de aplicaciones Web. El método de seguridad para controlar los accesos a una carpeta es el mismo que el que asegura la administración de Zope; esto da una idea de la potencia de la seguridad en las aplicaciones correctamente generadas con Zope.

Cuando un usuario intenta acceder a un recurso protegido, Zope obliga al usuario a facilitar un nombre de usuario y una contraseña que permita el acceso al recurso.

La seguridad, en general, lleva asociada dos conceptos: autentificación y autorización. Autentificación significa saber quién es el usuario y autorización significa determinar que está haciendo. Zope proporciona facilidades separadas para manejar el proceso de autentificación de los usuarios y la administración de los permisos que les permiten o impiden realizar acciones.

3.6.3.5.1.1 Autentificación.

Si no existe un producto que gestione la autentificación, el aspecto de la ventana de autentificación dependerá del sistema operativo sobre el que se esté ejecutando Zope. Por ejemplo, con Windows XP tiene el aspecto de la Figura 3.20.

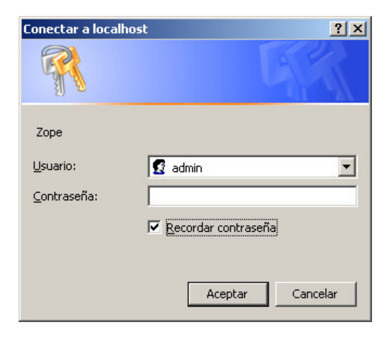


Figura 3.20. Autentificación de usuarios en Zope.

Cuando se introducen los datos de una cuenta de usuario correctamente, Zope comprueba los permisos del usuario sobre el objeto, permitiendo o denegando la operación que se pretenda realizar.

Las cuentas de usuario se manejan en objetos llamados *Carpetas de Usuario* (User Folder). Este objeto es una carpeta especial que sólo permite incluir usuarios. Debe ser única por cada carpeta (Folder) del servidor. Zope la nombra como *acl_users*.



Figura 3.21. Vista de una carpeta de usuarios (User Folder)

Como puede verse en la Figura 3.21, el interfaz de una carpeta de usuarios permite añadir nuevos usuarios y modificar y eliminar usuarios existentes. Para añadir un nuevo usuario y para modificar sus propiedades, el formulario que se debe rellenar es el mismo. Hay que facilitar el nombre de usuario (el identificador no puede repetirse en la carpeta), la contraseña (dos veces, para evitar fallos al teclearla), el dominio (lista opcional de dominios en los que el usuario tiene permitido el acceso; si se deja en blanco no existen restricciones) y los roles (este concepto se explica más adelante).

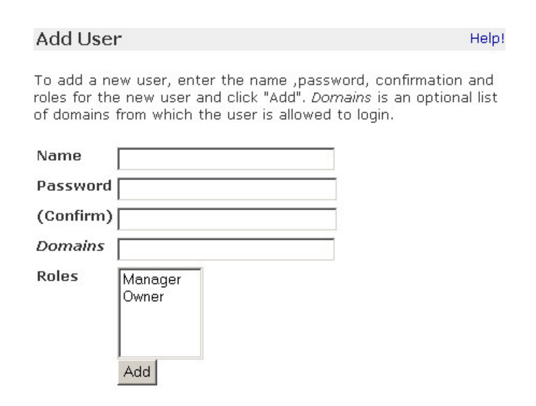


Figura 3.22. Información de una cuenta de usuario Zope.

En Zope pueden existir múltiples *carpetas de usuario* situadas en diferentes *carpetas* de la jerarquía del servidor. Un usuario no puede acceder a recursos que están

situados por encima de su *carpeta de usuario.* Sólo los usuarios definidos en la *carpeta de usuario* situada en la carpeta raíz (*root*) del servidor, tienen posibilidad de acceder a toda la jerarquía de objetos del servidor. Tras instalar Zope, el usuario elegido para administrarlo es el único existente y se almacena en la *carpeta de usuario* de la *carpeta* raíz. Esto permite definir una simple, pero potente a la vez, política de seguridad.

Si existen departamentos, secciones, etc. independientes en una organización, empresa, negocio, etc, creando una *carpeta* para cada departamento (por ejemplo, ventas y contabilidad) e incluyendo una *carpeta de usuario* para cada uno de ellos, automáticamente se asegura que los usuarios de un departamento no tendrán acceso al otro y viceversa. Y más importante aún, si se añaden en cada *carpeta de usuario* uno o varios usuarios con rol de administrador (Manager), este usuario será administrador de ese departamento o parte de la jerarquía del servidor. Este patrón de seguridad en Zope se conoce como delegación (*delegation*). Delegando la responsabilidad de administración en diferentes áreas de la jerarquía se consigue transformar un problema considerable en varios más pequeños, con las ventajas que esto conlleva en Ingeniería del Software.

3.6.3.5.1.2 Autorización y políticas de seguridad.

Zope controla la autorización mediante políticas de seguridad. Éstas definen qué puede hacer cada usuario. Un concepto importante en la autorización es el *rol*. Los roles permiten definir clases de usuarios y permisos que protegen los objetos. De esta forma, las políticas de seguridad permiten definir qué roles (clases de usuarios) pueden hacer qué acciones (permisos) en una parte determinada de la jerarquía de objetos Zope. El uso de roles facilita la administración de permisos.

Existen roles predefinidos en Zope:

- *Manager:* este rol permite realizar todas las funciones estándar de administración de Zope. Dicho de otra forma, no tiene limitaciones.
- *Anonymous:* es un rol que únicamente tiene el usuario predefinido `anonymous user` que es todo usuario que no se ha autentificado en el sistema. Permite visualizar objetos públicos, nunca modificarlos ni eliminarlos.

- *Owner:* se asigna automáticamente a un usuario sobre el contexto en el que crea un objeto.
- *Authenticated:* se asigna automáticamente a los usuarios que facilitan credenciales de validación correctas al sistema zope. Significa que Zope sabe que es un usuario del sistema.

Es posible crear nuevos roles. Cuando en el marco derecho tenemos el contenido de una carpeta, en la parte superior existe una pestaña etiquetada como `Security`. Al pulsar sobre ella, entre otras cosas que se comentarán más adelante, puede observarse la lista de roles existente en ese contexto.



Figura 3.23. Lista inicial de roles en Zope.

Haciendo *scrolling* hasta la parte inferior del marco derecho, se visualiza un formulario para la creación y el borrado de roles. Para la creación basta con dar el nombre elegido.

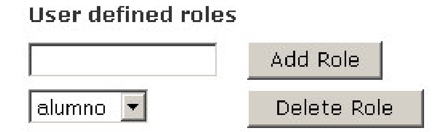


Figura 3.24. Añadir y eliminar rol.

Es importante destacar que cada rol tiene efecto en el contexto en que está definido.

Finalmente, para gestionar la políticas de seguridad Zope define una lista de permisos aplicables a tipo de objeto. En la pestaña *Security* se visualiza la relación de permisos aplicables a cada objeto.

Los permisos definen qué acciones se pueden hacer sobre los objetos Zope. La tabla presenta una casilla de verificación en cada intersección permiso-rol cuyo significado es el siguiente: si la casilla está activada, los usuarios que posean el rol, pueden realizar las acciones que protege el permiso, en otro caso no (a no ser que se tenga el permiso por adquisición como se verá más adelante).

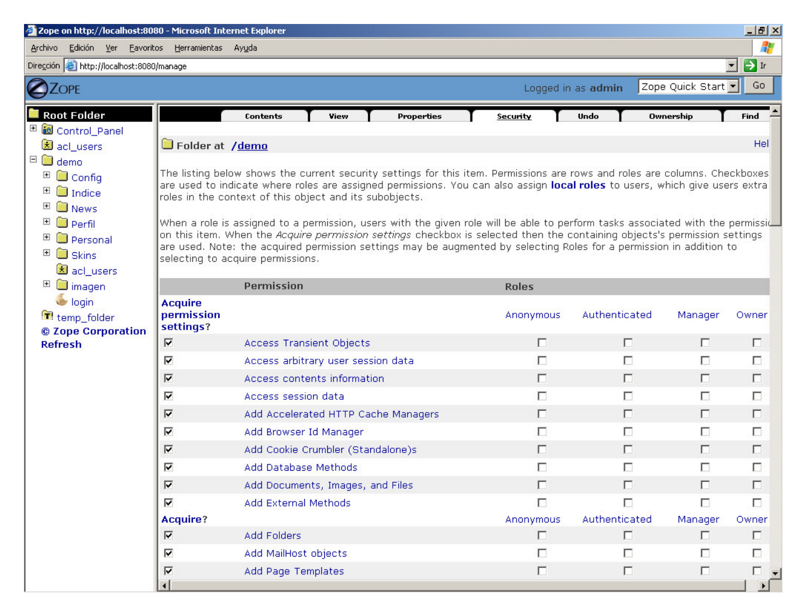


Figura 3.25. Lista de permisos Zope.

Cuando se instala Zope se instancian los permisos de la *carpeta* raíz. Al rol *Manager* se le asignan todos los permisos mientras que al rol *Anonymous* se le asignan permisos que garanticen la seguridad e integridad del sistema.

A todos los objetos creados se le asigna la política de seguridad de *adquisición*. Esto significa, que se le asignan los mismos permisos que tenga la *carpeta* que contiene el objeto recién creado. Visualmente puede comprobarse como para cada permiso se activa la casilla de verificación de la columna *Adquire permission settings?* y se desactivan todas las intersecciones rol-permiso.

Acquire permission settings?	Permission	Roles: Anonymous	Authenticated	Manager	Owner
☑	Access contents information	☐	☐	☐	☐
☑	Change permissions	☐	☐	☐	☐
☑	Copy or Move	☐	☐	☐	☐
☑	Delete objects	☐	☐	☐	☐
☑	Manage WebDAV Locks	☐	☐	☐	☐
☑	Manage properties	☐	☐	☐	☐
☑	Manage users	☐	☐	☐	☐
☑	Take ownership	☐	☐	☐	☐
☑	Undo changes	☐	☐	☐	☐
☑	View	☐	☐	☐	☐

Figura 3.26. Adquisición de permisos del contenedor.

Para modificar esta política por defecto, se debe desactivar la casilla de adquisición del permiso que se pretenda modificar y activar la casilla correspondiente al rol al que se le desee conceder el permiso.

3.6.3.6.- *Conectividad con bases de datos relacionales.*

Zope permite realizar conexiones a bases de datos relacionales externas. Las conexiones *(Database Connection)* deben crearse antes de definir los métodos de manipulación y administración de datos. El motivo es que cada método SQL *(Z SQL Method)* está asociado a una conexión. Zope dispone de adaptadores para la mayoría de sistemas gestores de bases de datos relacionales: Oracle, Sybase, Interbase, ODBC, PostgreSQL, MySQL, etc. Además, incluye un adaptador para Gadfly, una base de datos relacional escrita en Python con propósitos de demostración, ya que sólo es aconsejable su utilización con volúmenes de datos pequeños.

Para crear una conexión se debe instalar previamente el adaptador correspondiente. Como ejemplo, se muestra cómo crear una conexión a una base de datos `.mdb` mediante el adaptador ODBC. El primer paso es crear el origen de datos apuntando a la base de datos (este proceso está al margen de Zope y deben utilizarse las utilidades que facilita el sistema operativo).

De la lista de objetos se selecciona *Z ODBC Database Connection* y se rellena el formulario facilitando el identificador y el título de la conexión, el origen de datos anteriormente creado o la cadena de conexión y si se desea realizar la conexión inmediatamente.

Add Z ODBC Database Connection

Id ODBC_database_connection
Title Z ODBC Database Connection
Select an ODBC Data Source
MS Access Database, Microsoft Access Driver (*.mdb)
dBASE Files, Microsoft dBase Driver (*.dbf)
Excel Files, Microsoft Excel Driver (*.xls)
mio, Microsoft Access Driver (*.mdb)
aljada, Microsoft Access Driver (*.mdb)
demo, Microsoft Access Driver (*.mdb)
or enter a Database Connection String [1]
Connect immediately ☑
Add

1 The database connection string is of the form:

```
dsn user password
```

where the data source name, user id, and password are separated by one or more spaces.

Figura 3.27. Crear una conexión a una base de datos relacional externa.

Para ejecutar código SQL, se puede hacer tecleándolo directamente en la pestaña *Test* de la conexión recién creada o creando un objeto `SQL Method` (recomendable para depurar errores y sobre todo, para la reutilización).

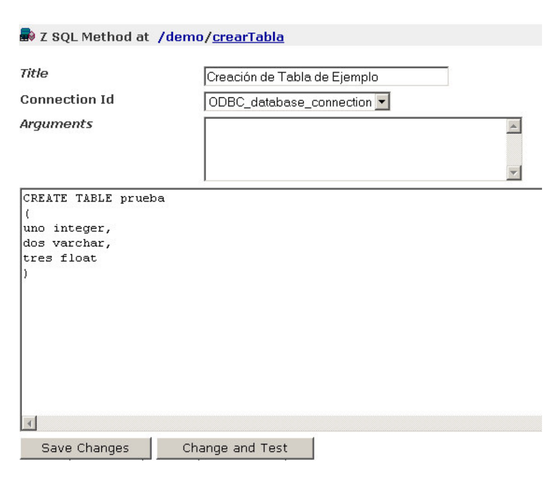


Figura 3.28. Ejecutar código SQL sobre una base de datos relacional externa.

Si posteriormente se desea añadir registros se podría crear un nuevo método SQL con el siguiente código:

```
insert into prueba values (1,'uno',1.10);
insert into prueba values (2,'dos',2.20);
insert into prueba values (3,'tres',3.30);
```

Pero, este método de ejecutar SQL no tiene mucha utilidad en desarrollo de aplicaciones, ya que lo normal es que los datos a insertar se decidan en tiempo de ejecución. La solución es utilizar métodos SQL dinámicos. Los datos se envían a través de un formulario. Un ejemplo de código:

```
<html>
<body>
<p align="center">Introducir datos de prueba</p>
<form method="POST" action="insertar”>
  <p align="center">uno <input type="text" name="uno"></p>
  <p align="center">dos<input type="text" name="dos"></p>
  <p align="center">tres <input type="text" name="tres"></p>
  <p align="center"><input type="submit" value="Insertar"></p>
</form>
</body>
</html>
```

y posteriormente crear un método SQL llamado `insertar` cuyo código sea:

```
insert into prueba values (
<dtml-var unotexto>,'<dtml-var dos>',<dtml-var tres>)
```

En la creación del método es necesario incluir en el área de texto para argumentos (ver Figura 3.28) los nombres de los argumentos recibidos por el método: `uno`, `dos`, `tres`.

Finalmente se muestra como tratar el resultado de una consulta realizada con la cláusula SQL `select`. Si se crea un método SQL llamado `listarRegistros` cuyo código sea:

```
select * from prueba
```

un documento DTML que visualiza el resultado podría ser el siguiente:

```
<html>
<body>
<table>
    <tr>
      <td>uno</td>
      <td>dos</td>
      <td>tres</td>
    </tr>
<dtml-in listarRegistros>
    <tr>
      <td><dtml-var uno></td>
      <td><dtml-var dos></td>
      <td><dtml-var tres></td>
    </tr>
</dtml-in>
  </table>
</body>
</html>
```

3.6.3.7.- *Crear nuevos productos.*

La comunidad de usuarios y desarrolladores de Zope pueden crear y distribuir nuevos productos (objetos). Existen cientos de productos desarrollados que realizan tareas específicas y que se distribuyen desde http://www.zope.org/Products.

Existen dos formas de desarrollar productos Zope: a través del ZMI usando ZClasses y utilizando el lenguaje de programación Python. También es posible un método híbrido. La forma más sencilla de hacerlo es usando ZClasses.

Los productos están situados en la carpeta `Product Management Folder` del panel de control. Para crear un nuevo producto se debe pulsar el botón `Add Product` y facilitar el identificador (`PortalIES`) y el título (`Portal Genérico IES`) y pulsar el botón `Generate`.



Figura 3.29. Un producto recién creado.

El siguiente paso es crear el formulario que se visualizará para crear un objeto de la lista facilitada por Zope. Por ejemplo, un método DTML llamado `addForm`.

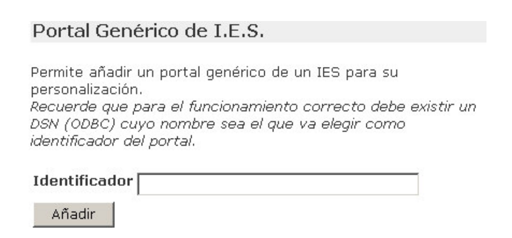


Figura 3.30. Formulario que rellenará el usuario para crear un nuevo producto.

El siguiente paso es crear un objeto *factoría (Zope Factory)* facilitándole la siguiente información: identificador (`FactPortal`), el título (`Portal Generico`), el nombre del objeto que aparecerá en la lista Zope (`Portal Generico de IES`) y el método al que se invocará cuando se realice una instancia del objeto (`addForm`).

Finalmente, se debe crear el programa que realice la instancia del objeto nuevo a partir del formulario rellenado por el usuario. Aunque existen varias posibilidades de desarrollar productos, como se vió anteriormente, la más sencilla consiste en copiar el objeto desarrollado vía Web con el ZMI (`portalgen`) y hacer un *script* (`add`, suponiendo que así se nombró en el formulario `addForm`) que copie ese objeto y lo renombre con el identificador que el usuario haya decidido. La Figura 3.31 muestra como quedaría el producto recién creado (la carpeta `Help` se crea por defecto).

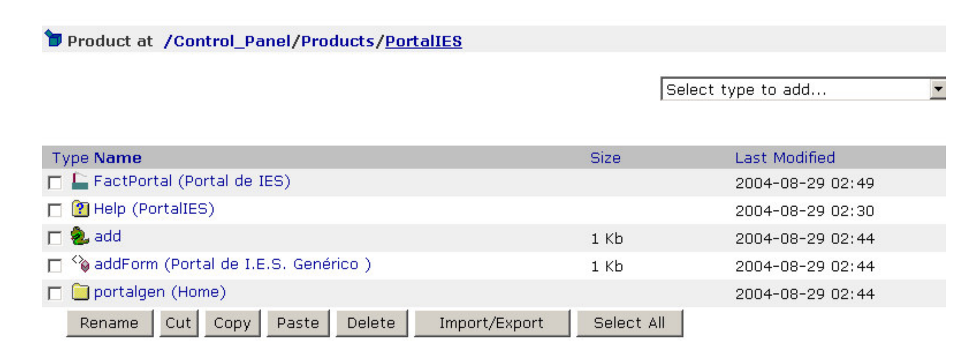


Figura 3.31. Contenido de un producto nuevo.

El *script* Python llamado `add` que se encarga de instanciar el nuevo objeto tendría un código fuente similar al siguiente:

```
#El script recibe un parámetro llamado id con el identificador que el
#usuario ha elegido
#Paso 1. Clonar la carpeta portalgen que contiene el objeto a instanciar.

exhibit=context.manage_clone(container.portalgen,id)

#Paso2. Poner título al nuevo objeto
exhibit.manage_changeProperties(title='Home')

#Paso 3. Si la creación del nuevo producto es vía Web, redireccionar al #contexto llamador

if REQUEST is not None:
 try:
  u=context.DestinationURL()
 except:
  u=REQUEST['URL1']

 REQUEST.RESPONSE.redirect(u+'/manage_main?update_menu=1')
```

Finalmente, como el *script* anterior crea un nuevo objeto en el contexto deseado se necesita cambiar el permiso al *objeto factoría* a `Add Documents, Images and Files`. A partir de este momento, cuando se intente añadir un nuevo objeto en cualquier carpeta aparecerá un nuevo tipo de objeto `Portal Generico de IES`.

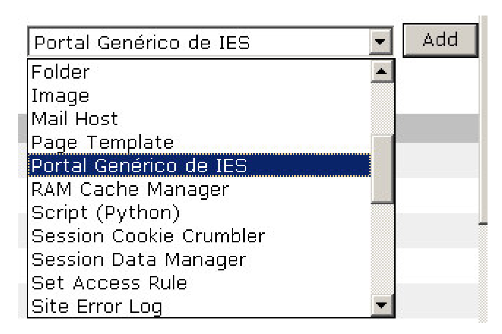


Figura 3.32. El producto creado pasa a formar parte de la lista de objetos Zope.

Si se desea poder distribuir el producto, hay que realizar un último paso. En la vista del producto, hay que pulsar la pestaña `Distribution` e introducir siguientes valores al formulario: la versión del producto y si se desea que el producto pueda ser modificado o no en la distribución. También existe la posibilidad de elegir qué objetos se permiten modificar y cuales no.

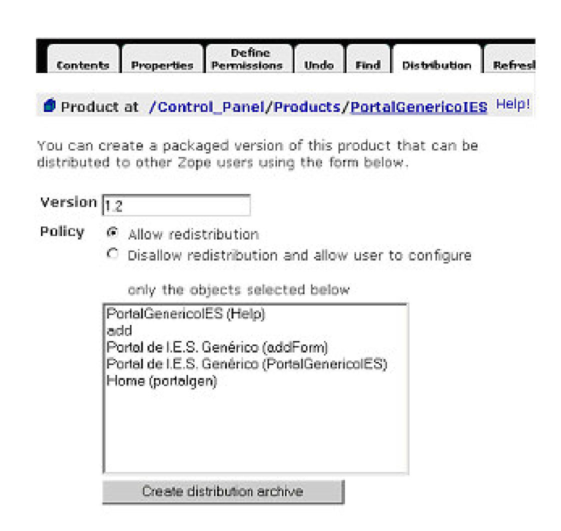


Figura 3.33. Distribución de un producto Zope.

El proceso de distribución genera un archivo cuyo nombre es el nombre del producto, un guión, la versión y la extensión `.tar.gz`. En el ejemplo seguido, el nombre del archivo generado por Zope sería `PortalIES-1.0.tar.gz`.

3.6.3.8.- *CMF (Content Management Framework).*

CMF es el gran proyecto Zope para la gestión de contenido. Permite la gestión y edición de documentos de manera colaborativa a través del propio portal que se está gestionando. Pueden introducirse varios tipos de páginas en el portal. Las primeras de ellas son las que presentan contenido público a los visitantes. Además eixten páginas personales de los usuarios, que son privadas y es donde guardan sus documentos de trabajo. Hay otras como la página de noticias en donde se publican todos los objetos de ese tipo que se crean en cualquier parte del portal. Las últimas noticias también aparecen en una columna de la página de inicio si así se desea.

Las principales características de un portal basado en CMF son:

- Permite la edición colaborativa de documentos.
- Presenta facilidades para el "log in" y "log out".
- El portal se estructura en carpetas en las que se almacenan los documentos publicados. En esas carpetas se pueden añadir nuevos archivos, editar los ya existentes, etc.
- Los documentos básicos con que cuenta el portal son: noticias, documentos, ficheros, imágenes, enlaces y carpetas.
- Cada documento lleva asociado unos meta-datos (título, tema, descripción) para una mejor catalogación e identificación. Además se guarda un historial de ellos para ver los cambios que se han realizado y deshacerlos si es preciso.
- El portal lleva incorporado un motor de búsqueda.
- Los usuarios disponen de páginas personales, pueden establecer sus preferencias (aspecto, correo electrónico,...) y crear una colección de favoritos para reunir los enlaces a los documentos que más usan.
- Se puede activar un sistema de flujo de trabajo (*workflow*) para controlar paso a paso el proceso de publicación de documentos.
- CMF también tiene un sistema de seguridad y control de usuarios que hereda directamente de Zope.
- La imagen de portal está controlada por una serie de "pieles" (*skins*). Existen varios y los usuarios pueden elegir el que más les guste en el menú de preferencias. Es posible y muy sencillo crear otros nuevos porque todo está controlado por CSS. Para hacer modificaciones más profundas de la presentación hay que editar una serie de plantillas (ZPT) y cambiar las imágenes del correspondiente archivo.
- Es posible añadir funciones extras a un portal CMF porque existe gran cantidad de productos desarrollados basados en él.

3.6.4.- Plone.

Plone es un generador de portales Web construido sobre la sólida base de Zope. Permite la creación, personalización y gestión de un sitio Web de manera rápida y fácil.

Todas las acciones que se han de realizar para la gestión de Plone se pueden realizar a través de un interfaz Web, lo que facilita el trabajo colaborativo y distribuido. Plone es un proyecto desarrollado por una amplia comunidad y su licencia es GPL. Se puede probar Plone sin necesidad de instalarlo en el sitio creado por el propio proyecto Plone para pruebas.

Tiene toda la potencialidad de CMF y destaca en particular por las siguientes características:

- Su menú de navegación: es una herramienta muy útil que muestra las diferentes opciones de navegación disponibles desde la página de inicio del portal y el camino que conduce a la página en que se está. Lo más interesante de este menú es que se construye automáticamente con los objetos por los que puede navegar el usuario que está accediendo; es decir, la vista del menú de navegación es distinta para usuarios con distintos permisos de acceso, y cuando se publica un nuevo objeto éste aparece automáticamente para los usuarios que están autorizados a verlo. Esto simplifica enormemente la navegación y el proceso de integración en el portal de la información publicada. Adicionalmente, una barra horizontal, que separa el encabezamiento de la página y el cuerpo, contiene en todo momento el camino desde la página de inicio hasta la página actual, siendo además practicable cada paso del camino. Con todo ello se logra una navegabilidad total de forma automática.
- El calendario de eventos: se muestra un calendario en el que cada usuario puede ver indicadas las fechas en que otros han publicado eventos. Esto facilita la coordinación distribuida de una agente colectiva.
- El sistema de noticias para anunciar novedades: cada usuario puede publicar una noticia en su propia carpeta, todo lo que tiene que hacer es caracterizarla correctamente con un título significativo, una descripción y un contenido correctos, y unas fechas de publicación y caducidad bien escogidas. El portal se encargará de insertar esa noticia una vez publicada en la página de noticias mientras esté vigente, y además las noticias más nuevas aparecerán si se quiere en una ranura lateral del portal.
- La posibilidad de integrar otros productos CMF.

- La internacionalización y localización. Estas características, que se han evaluado negativamente en otros CMS, dan muy buen resultado en Plone. Se pueden tener diferentes versiones de los objetos del portal en diferentes lenguas, y se puede hacer que Zope sirva la versión apropiada al cliente si este manifiesta alguna preferencia en el diálogo HTTP. Los catálogos de mensajes para localizar la interfaz se presentan de forma muy apropiada, facilitando la búsqueda y la traducción de mensajes que aún no han sido localizados. Lo que se obtiene finalmente es una interfaz adaptada a las preferencias del navegador y una versión del portal por defecto coherente con esa elección pero sin renunciar a la posibilidad de navegar de una versión a otra.
- La transparencia con que se definen consultas y subconsultas en jerarquías navegables similares a los directorios y subdirectorios y que permiten navegar por la información en función de su semántica y no de su posición en el portal.
- Es posible sindicar objetos, en particular foros o noticias, con RSS, y en ese caso el intercambio de información se produce en XML. También se puede obtener una versión XML o traducciones a otros formatos directamente a través de la interfaz de usuario, sobre todo a partir de la versión 2.0 de Plone.
- El proceso de flujo de trabajo (workflow) permite controlar muy bien el rango de visibilidad de un objeto, así como el nivel de privilegio que tiene que tener un usuario para provocar una transición de un estado a otro en un objeto. Los siguientes estados son posibles:
    - *Privado*: sólo el propietario del objeto y los administradores pueden verlo. No consta, ni a efectos de navegación ni a efectos de búsquedas, para el resto de los usuarios.
    - *Visible*: puede ser referenciado y alcanzado navegando, pero no aparece en el árbol de navegación. Sigue siendo editable por el propietario y los administradores.
    - *Pendiente*: cuando un objeto ya está bien según el criterio del propietario, éste querrá publicarlo. Si tiene los permisos suficientes, el propietario podrá optar a publicarlo él mismo, pero si es un simple miembro del portal podrá proponer su publicación a los revisores, y en ese transitorio el estado del objeto es pendiente. Tiene las mismas propiedades de un objeto visible y además se anuncia en la

interfaz de los revisores para que estos puedan proceder a su examen y publicación.

- *Publicado*: el objeto se pone a la disposición de todos los usuarios que tengan acceso a esa parte del portal y aparece para ellos en el menú de navegación. Deja de ser editable para el propietario, aunque sí puede revertir su estado a visible o a privado para editarlo. El administrador sí puede editarlo.
- *Público*: El objeto es visible para todo el mundo, es navegable y aparece en el menú de navegación de todos los usuarios. Además, es modificable por su propietario y los administradores sin tener que retirarlo.

Plone puede instalarse básicamente de dos formas: como portal privado o como portal público. En el primer caso, no se ofrece a los usuarios la posibilidad de suscribirse de forma autónoma, el estado por defecto de un nuevo objeto creado es privado y todos los estados existen. En el segundo caso, cualquiera puede convertirse a sí mismo en usuario autentificado del portal, el estado por defecto de un objeto es visible y no existe el estado público.

También hay varios tipos de usuarios (roles), con distintos permisos en cada parte de un portal:

- *Administrador*: tiene permiso total sobre objetos en cualquier estado, puede cambiar la propiedad de cualquier objeto, puede crear y destruir usuarios y modificar sus permisos.
- *Revisor*: es un usuario miembro con la potestad de publicar artículos que han sido sometidos a revisión por autores sin permisos para publicar por sí mismos.
- *Miembro*: es un usuario autentificado en el portal, y tiene derecho a ver los contenidos que están en cualquier estado menos *privado*.
- *Propietario*: un miembro es propietario de los objetos de los que es autor, pero también puede ser declarado propietario de otros objetos y comparte con su autor los derechos de edición y gestión en igualdad de condiciones.

El sitio Web del grupo Oreto está desarrollado con Plone.



Figura 3.34. Portal del grupo Oreto.

4.- MÉTODO DE TRABAJO.

El método de trabajo seguido para la realización del proyecto ha estado influido por las fases en las que se ha desarrollado:

a) Realización de un estudio profundo sobre el estado de la cuestión, incidiendo en el desarrollo de software de aplicaciones Web (metodologías, arquitecturas, tecnologías, etc.).
b) Elección de la metodología y la tecnología para resolver el problema y conseguir el objetivo principal de este proyecto fin de carrera.
c) Aprendizaje pormenorizado del servidor de aplicaciones Zope, del lenguaje Python y las posibilidades de la combinación.
d) Estudio y análisis de los requisitos de la aplicación a desarrollar.
e) Diseño de la aplicación.
f) Implementación de la aplicación.
g) Realizar el manual de usuario de la aplicación.
h) Puesta en funcionamiento de una instancia de GASPWIES para el I.E.S. Aljada de Murcia, que ha servido como producto prototipo para pruebas.

No se han numerado las fases porque no se han realizado secuencialmente. De hecho, algunas se han solapado en el tiempo. Otras se han realimentado a medida que avanzaba el proyecto.

El resultado de la fase a) se ha redactado en el apartado 1.

En la fase b) se tomó la decisión de utilizar el servidor de aplicaciones Zope, como base de la tecnología a utilizar.

Algunos de los conceptos aprendidos en la fase c) han sido reflejados en el apartado dedicado a los sistemas gestores de contenidos.

Las fases g) y h) se incluye en el apartado 5.

El resto de este apartado se dedica a la exposición de las fases d), e) y f).

En definitiva, a continuación se realiza un estudio del ciclo de vida seguido para el desarrollo de la aplicación. Debido a la extensión de su funcionalidad, se han elegido los

elementos claves para la comprensión de la funcionalidad, el diseño y los detalles de implementación de la aplicación.

4.1.- ESTUDIO Y ANÁLISIS REQUISITOS DE LA APLICACIÓN.

Este apartado realiza un estudio del ciclo de vida seguido para el desarrollo de la aplicación.

4.1.1.- Organización y funcionamiento de los Institutos de Enseñanza Secundaria.

Un Instituto de Enseñanza Secundaria (IES) es un Centro de educación de alumnos supervisado y sostenido económicamente por la Consejería de Educación de la Comunidad Autónoma en la que se encuentre situado.

Cada IES está autorizado para impartir determinadas enseñanzas que están legisladas por el Ministerio de Educación y Cultura. Según el ámbito del IES, el número de enseñanzas impartidas será mayor o menor.

Todos los Centros imparten Educación Secundaria Obligatoria (primer ciclo, segundo ciclo o ambos) cuyos alumnos tienen edades comprendidas entre 12 y 16 años.

La mayoría de los Centros imparten Bachillerato (en alguna, varias o todas sus modalidades) para alumnos mayores de 16 años que han obtenido el título en ESO. Actualmente existen cuatro modalidades de Bachillerato: Humanidades y Ciencias Sociales, Ciencias de la Salud, Tecnología y Bachillerato Artístico.

Algunos Centros imparten enseñanzas de Formación Profesional Específica. Estas enseñanzas se dividen en familias profesionales (Informática, Electricidad, Sanidad, Administración, y un largo etcétera) que tienen competencia para impartir Ciclos Formativos. Los Ciclos Formativos pueden ser de Grado Medio, para alumnos titulados en ESO y pueden ser de Grado Superior, para alumnos con el título de Bachillerato.

Existen otras enseñanzas regladas que se imparten en los Centros de forma excepcional como son la Garantía Social, Educación de Adultos, etc.

En los IES existe un órgano llamado Consejo Escolar que aprueba todas las actuaciones propuestas por la Comunidad Educativa. Está presidido por el Director del Centro y sus miembros representa al claustro de profesores, a padres y madres, a alumnos, al personal no docente y al Ayuntamiento en el que está ubicado el Centro.

Cada IES tiene un documento llamado Proyecto Educativo de Centro que refleja los aspectos más importantes del Centro como son las señas de identidad, el entorno socio-económico, criterios generales para consecución de objetivos en etapas, reglamento de régimen interno, etcétera.

Los miembros con mayor influencia en la vida del Centro son los alumnos y el claustro de profesores. También los padres y madres de alumnos y el personal no docente forman parte de la Comunidad Educativa.

En todos los IES existe un órgano de gobierno llamado Equipo Directivo que está compuesto por el Director, el Secretario, el Jefe de Estudios y varios Jefes de Estudios Adjuntos (de ESO, de Bachillerato y/o de Formación Profesional).

Las enseñanzas están organizadas en Departamentos Didácticos. Cada Departamento tiene competencias sobre determinadas materias. Todos los profesores del Centro están adscritos a un único Departamento Didáctico, incluidos los profesores que forman el Equipo Directivo. Para cada Departamento, existe un profesor coordinador cuyo cargo se denomina Jefe de Departamento.

Cada Departamento Didáctico tiene un documento que contiene las programaciones didácticas de cada una de las materias de las que tiene competencia. En la programación didáctica se reflejan, al menos, los objetivos, contenidos, metodología y criterios de evaluación y recuperación de la materia en cuestión.

Los profesores imparten una o varias materias a uno o varios grupos de alumnos en una o varias etapas. Las materias se denominan asignaturas en el caso de ESO y Bachillerato y módulos profesionales en el caso de la Formación Profesional.

Una de sus funciones consiste en evaluar la consecución de los objetivos propuestos en las programaciones didácticas de las materias por parte de los alumnos. El proceso de evaluación se divide en tres evaluaciones y una calificación final. En cada evaluación el profesor realiza las pruebas que estime oportunas.

4.1.2.- Entrevista con el cliente.

Esta fase comienza con reuniones a las que asinsten profesores de distintos Departamentos Didácticos a los que les interesa que su Centro tenga presencia en Internet con un portal en el que participen de forma activa todos los miembros de la Comunidad Educativa que lo deseen.

Las conclusiones más importantes extraídas en diferentes reuniones sobre la funcionalidad del sitio Web son las siguientes:

- Los miembros del Equipo Directivo serán los que tengan la responsabilidad de "administración" y configuración del sitio Web. Aunque si así lo desean podrán delegar la responsabilidad en otro profesor o grupo de profesores.
- Habrá una parte de la información incluida en el portal que será estática (la historia, las instalaciones, el proyecto educativo, etc.). Es decir, no cambiará a lo largo del curso. En cualquier caso, deberá facilitarse la modificación de este contenido.
- Se incluirá la relación del claustro de profesores con indicación, al menos, de su cargo, su horario de tutoría para padres y la forma de contactar con cada profesor.
- Será imprescindible que la responsabilidad de publicación no recaiga en una o dos personas, sino que quien lo desee pueda hacerlo.
- Las publicaciones estarán divididas en secciones y algunas podrán considerarse interesantes para ser publicadas como novedades del Centro.
- Deberá existir un mecanismo que permita diferenciar entre noticias públicas y noticias que sólo puedan visualizar el claustro de profesores.
- Las publicaciones deben permitir llevar documentos adjuntos.
- Los alumnos también podrán publicar información mediante sus representantes (delegados, representantes de alumnos en el Consejo Escolar, responsables de

actividades extraescolares, etc.) en una sección dedicada a ellos. En ningún caso las noticias publicadas por los alumnos podrán formar parte de las novedades.

- El flujo de información parte del Equipo Directivo hacia los Jefes de Departamento (en reuniones de comisión pedagógica) que transmiten su contenido a los profesores miembros (en reuniones de Departamento).
- Los Jefes de Departamento jugarán un papel importante. Se encargarán de mantener una sección dedicada a su Departamento Didáctico en la que se publicarán tanto las programaciones didácticas de las materias en las que tiene competencia el Departamento, como información de interés con origen o destino de los miembros del Departamento (profesores y alumnos).
- El Departamento de Actividades Complementarias y Extraescolares juega un papel muy relevante en la actividad del Centro por lo que deberá tener fácil acceso.
- En la gestión del Centro existe un elemento que resulta la piedra angular de la mecanización de los expedientes de los alumnos: el IES2000. El IES2000 es una aplicación informática que las Consejerías de Educación de las Comunidades Autónomas facilitan a los Centros para la gestión de matrículas, expedientes, datos del profesorado, etc. Esta aplicación almacena la información en una base de datos de Microsoft Access.
- En ningún caso existirá información personal en la Web (direcciones y teléfonos fundamentalmente).
- Resultaría interesante que los profesores hicieran públicas sus fichas de alumno. De esta manera, los padres que lo desearan podrían tener información actualizada de la trayectoria de sus hijos. Algunos padres, por motivos de trabajo, de distancia y otros, tienen dificultades para entrevistarse con los profesores de sus hijos. Cuando se entregan los boletines de notas al finalizar cada trimestre, algunos padres se llevan sorpresas desagradables.
- Los profesores que lo deseen podrán publicar información de interés para sus alumnos (avisos, hojas de ejercicios, soluciones, fechas de exámenes, etc.).
- Un miembro del equipo directivo propone la posibilidad de que exista un mecanismo de comunicación rápido para "*ahorrar tiempo, papel y gasto telefónico*".

- Se comenta la posibilidad de que exista una ayuda en línea en el portal, ya que algunos de los usuarios no son usuarios acostumbrados al uso de las nuevas tecnologías.

Esta lista se ha considerado como los requisitos funcionales de la aplicación (no existen requisitos no funcionales que merezca la pena mencionar).

Con posterioridad se celebró una nueva reunión para tratar el aspecto de la presentación y estructura del portal. Las conclusiones más importantes extraídas fueron las siguientes:

- Cada Centro suele tener sus colores insignia que normalmente están incluidos en su logotipo. Por este motivo, sería deseable que cada portal aporte su logotipo y pueda configurar el colorido del contenido.
- Aunque exista la posibilidad de que el interfaz incluya gráficos, es aconsejable permitir que se pueda reducir el número de elementos gráficos para mejorar la rapidez de acceso.
- Las opciones personales de los miembros del portal no se deben mezclar con los elementos de navegación del usuario anónimo.
- Las listas de noticias deben tener un tamaño limitado y, si es posible, configurable por el usuario.
- El acceso a las opciones debe ser intuitiva. Podría incluirse la posibilidad de que los usuarios personalizaran una lista de acceso rápido que incluyera las tareas más frecuentes que realiza.

De estas conclusiones se han extraído los requisitos estructurales y de navegación.

4.1.3.- Roles.

Antes de hacer un estudio de los casos de uso de la aplicación Web desarrollada, y en función de las conclusiones extraídas del apartado anterior, se considera que los usuarios estarán divididos en los siguiente grupos (roles):

a) *Usuarios anónimos*: son los visitantes que no están registrados en el portal. Podrán ser o no miembros de Comunidad Educativa.

b) *Alumnos*: serán los representantes de los alumnos del Centro.

c) *Profesores*: serán los miembros del claustro de profesores del Centro.

d) *Jefes de Departamento*: serán los profesores que tenga el cargo de Jefe de Departamento. Habrá uno por Departamento Didáctico. También son profesores.

e) *Equipo directivo*: serán el grupo de profesores que en cada momento formen parte del Equipo Directivo del Centro. También son profesores.

f) *Administrador*: es un usuario especial que permite iniciar la fase de configuración, personalización y administración del portal. Será el único usuario existente inicialmente.

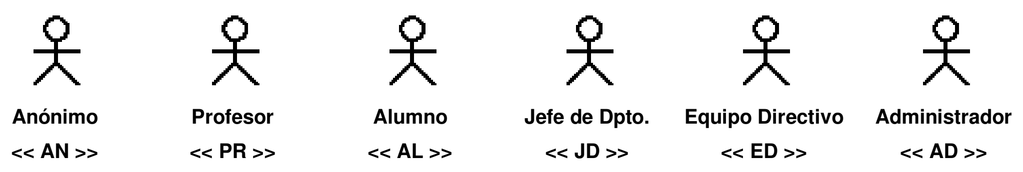


Figura 4.1. Roles de interés para la aplicación GASPWIES.

4.1.4.- Modelo de requisitos.

Si bien la fase de recogida de requisitos está dividida en la realización de los modelos de casos de uso, de tareas y de usuario, y con el objetivo de simplificar la visión general de la aplicación, se opta por realizar el diagrama de casos de uso inicial y una tabla-esquema con las tareas identificadas (algunas se dividen en subtareas como se explica más adelante), los usuarios implicados y los casos de uso asociados, para posteriormente desarrollar aquellos casos de uso y/o tareas que resulten de especial interés. La numeración de los casos de uso se realiza con el objetivo de simplificar el diagrama inicial de casos de uso mostrado en la Figura 4.2.

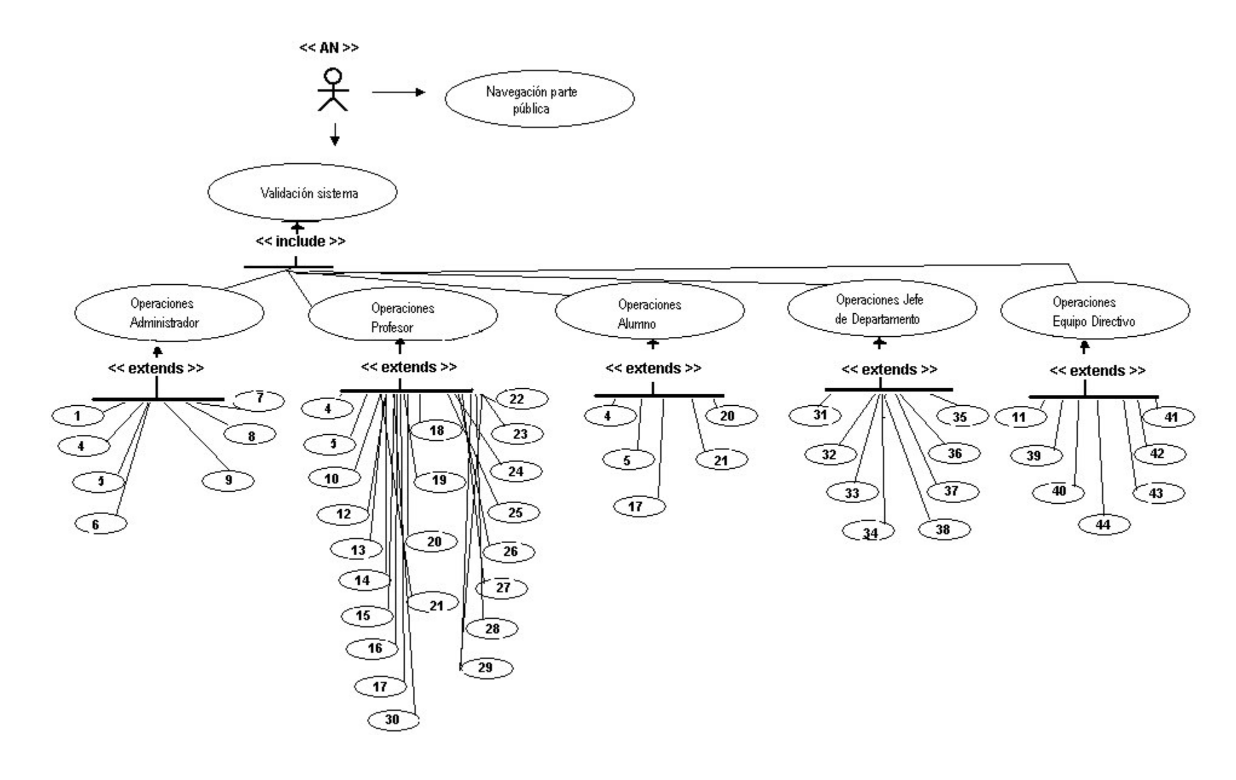


Figura 4.2. Diagrama inicial de casos de uso.

CASO DE USO	TAREA/S	ROLES
1. Activar el portal.	Activar el portal.	AD
2. Navegación en la parte pública	Navegación en la parte pública	AN, AD, PR, AL
3. Validación en el sistema	Validación en el sistema	AN
4. Desconexión del sistema	Desconexión del sistema	AD, PR, AL
5. Cambiar contraseña	Cambiar contraseña	PR, AL, AD
6. Configurar presentación	- Añadir logotipos - Añadir imagen de portada del Centro - Personalizar contenido público - Personalizar interfaz de usuario	AD
7. Añadir contenido público	- Publicar historia del centro. - Publicar enseñanzas impartidas. - Publicar el Proyecto Educativo del	AD

CASO DE USO	TAREA/S	ROLES
	Centro - Publicar la composición de los miembros del Equipo Directivo.	
8. Establecer parámetros de funcionamiento	Establecer parámetros de funcionamiento	AD
9. Definir usuarios que pertenecen al Equipo Directivo.	- Añadir usuario con rol de Equipo Directivo (Definir usuario maestro) - Establecer rol de Equipo Directivo para profesores del Centro. - Quitar rol de Equipo Directivo a profesores del Centro.	AD
10. Añadir imágenes de las instalaciones del Centro	Añadir imágenes de las instalaciones del Centro	PR
11. Eliminar imágenes del Centro	Eliminar imágenes del Centro	ED
12. Añadir grupos temáticos de información	Añadir grupos temáticos de información	PR
13. Añadir enlaces de interés de información	Añadir enlaces de interés de información	PR
14. Visualizar noticias con documentos adjuntos de interés para el profesorado	Visualizar noticias con documentos adjuntos de interés para el profesorado	PR
15. Agregar tarea a favoritas	Agregar tarea a favoritas	PR
16. Modificar lista de tareas favoritas	Modificar lista de tareas favoritas	PR
17. Enviar mensajes instantáneos	Enviar mensajes instantáneos	PR, AL
18. Consultar buzón de mensajes instantáneos	Consultar buzón de mensajes instantáneos	PR
19. Gestionar buzón de mensajes instantáneos	- Marcar mensajes como leídos - Borrar mensajes leídos	PR
20. Publicar noticias en el portal	- Publicar noticia - Adjuntar archivo a la noticia	AL, PR
21. Gestionar lista de noticias	- Visualizar lista propia de noticias - Modificar noticia publicada - Eliminar noticia publicada	AL, PR
22. Modificar datos personales	- Modificar fotografía personal - Modificar datos personales	PR

CASO DE USO	TAREA/S	ROLES
23. Gestionar materias impartidas	- Añadir materia. - Eliminar materia - Publicar contenido para materia	PR
24. Gestionar fichas de alumnos	- Preparar fichas de alumnos - Añadir ficha de alumno nuevo	PR
25. Definir pruebas a incluir en fichas para materias	Definir pruebas a incluir en fichas para materias	PR
26. Asignar calificaciones	Asignar calificaciones	PR
27. Realizar anotaciones una ficha de alumno	Realizar anotaciones una ficha de alumno	PR
28. Visualizar resumen de materia	Visualizar resumen de materia	PR
29. Visualizar ficha de alumno	Visualizar ficha de alumno	PR
30. Visualizar lista de usuarios y contraseñas de un grupo de alumnos	Visualizar lista de usuarios y contraseñas de un grupo de alumnos	PR
31. Realizar una publicación para página de Departamento	Realizar una publicación para página de Departamento	JD
32. Gestionar la lista de miembros de un Departamento	Gestionar la lista de miembros de un Departamento	JD
33. Registrar profesores en el portal	Registrar profesores en el portal	JD
34. Eliminar profesores del portal	Eliminar profesores del portal	JD
35. Registrar alumnos en el portal	Registrar alumnos en el portal	JD
36. Eliminar alumnos del portal	Eliminar alumnos del portal	JD
37. Modificar contraseña de alumno	Modificar contraseña de alumno	JD
38. Inicializar contenido de la página del Departamento	Inicializar contenido de la página del Departamento	JD
39. Añadir Departamentos Didácticos	Añadir Departamentos Didácticos	ED
40. Eliminar Departamentos Didácticos	Eliminar Departamentos Didácticos	ED
41. Añadir Jefe de Departamento	Añadir Jefe de Departamento	ED

CASO DE USO	TAREA/S	ROLES
42. Eliminar Jefes de Departamento	Eliminar Jefes de Departamento	ED
43. Generar contraseñas de alumnos	Generar contraseñas de alumnos	ED
44. Eliminar información obsoleta en el portal	- Eliminar mensajes antiguos y leídos - Eliminar fichas de alumnos del curso anterior	ED

El siguiente paso consiste en la realización de una ficha para cada tarea de las anteriores. Se muestran algunas que sirven como ejemplo.

TAREA: Activar el portal.	
Caso de uso:	Activar el portal
Objetivo:	Establecer el sitio Web como operativo.
Precondiciones:	Ninguno.
Situación de éxito:	Introducción correcta de todos los datos.
Situación de fracaso:	Falta algún dato por rellenar. En los campos numéricos se introducen caracteres alfanuméricos.
Tareas asociadas:	Validación en el sistema.

TAREA: Personalizar interfaz de usuario.	
Caso de uso:	Personalizar interfaz de usuario..
Objetivo:	Establecer el aspecto que tendrá el sitio Web.
Precondiciones:	Ninguna.
Situación de éxito:	Todos los casos.
Situación de fracaso:	En ningún caso.

TAREA: Personalizar interfaz de usuario.	
Tareas asociadas:	Esta tarea lleva asociadas varias subtareas consecuencia de un refinamiento del caso de uso: - Cambiar colorido: su objetivo es establecer los colores predominantes en la navegación y en la presentación de la información elegidos de las propuestas del sistema. - Añadir configuración de colorido. - Eliminar configuración de colorido: su objetivo es eliminar configuraciones personalizadas añadidas con anterioridad. - Elección de modo de navegación: texto o gráfico. - Personalizar iconos de navegación: su objetivo es añadir un juego nuevo iconos e imágenes de navegación si la configuración de navegación es con gráficos.

TAREA: Consultar ficha de alumno	
Caso de uso:	Personalizar interfaz de usuario..
Objetivo:	Establecer el aspecto que tendrá el sitio Web.
Precondiciones:	Ninguna.
Situación de éxito:	Todos los casos.
Situación de fracaso:	En ningún caso.

TAREA: Consultar ficha de alumno	
Tareas asociadas:	Esta tarea lleva asociadas varias subtareas consecuencia de un refinamiento del caso de uso: - Cambiar colorido: su objetivo es establecer los colores predominantes en la navegación y en la presentación de la información elegidos de las propuestas del sistema. - Añadir configuración de colorido. - Eliminar configuración de colorido: su objetivo es eliminar configuraciones personalizadas añadidas con anterioridad. - Elección de modo de navegación: texto o gráfico. - Personalizar iconos de navegación: su objetivo es añadir un juego nuevo iconos e imágenes de navegación si la configuración de navegación es con gráficos.

Algunas de las tareas del modelo de tareas pueden considerarse como flujos de trabajo. Las Figuras 4.3, 4.4, 4.5 y 4.6 muestran algunas de ellas.



Figura 4.3. Flujo de trabajo para publicar una noticia.

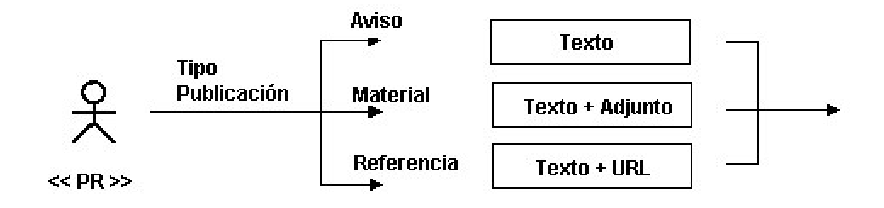


Figura 4.4. Flujo de trabajo para publicar contenido para página de materia.



Figura 4.5. Flujo de trabajo para añadir un Jefe de Departamento.

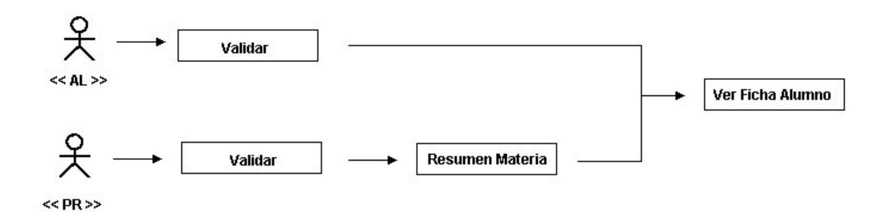


Figura 4.6. Flujo de trabajo para visualizar ficha de alumno.

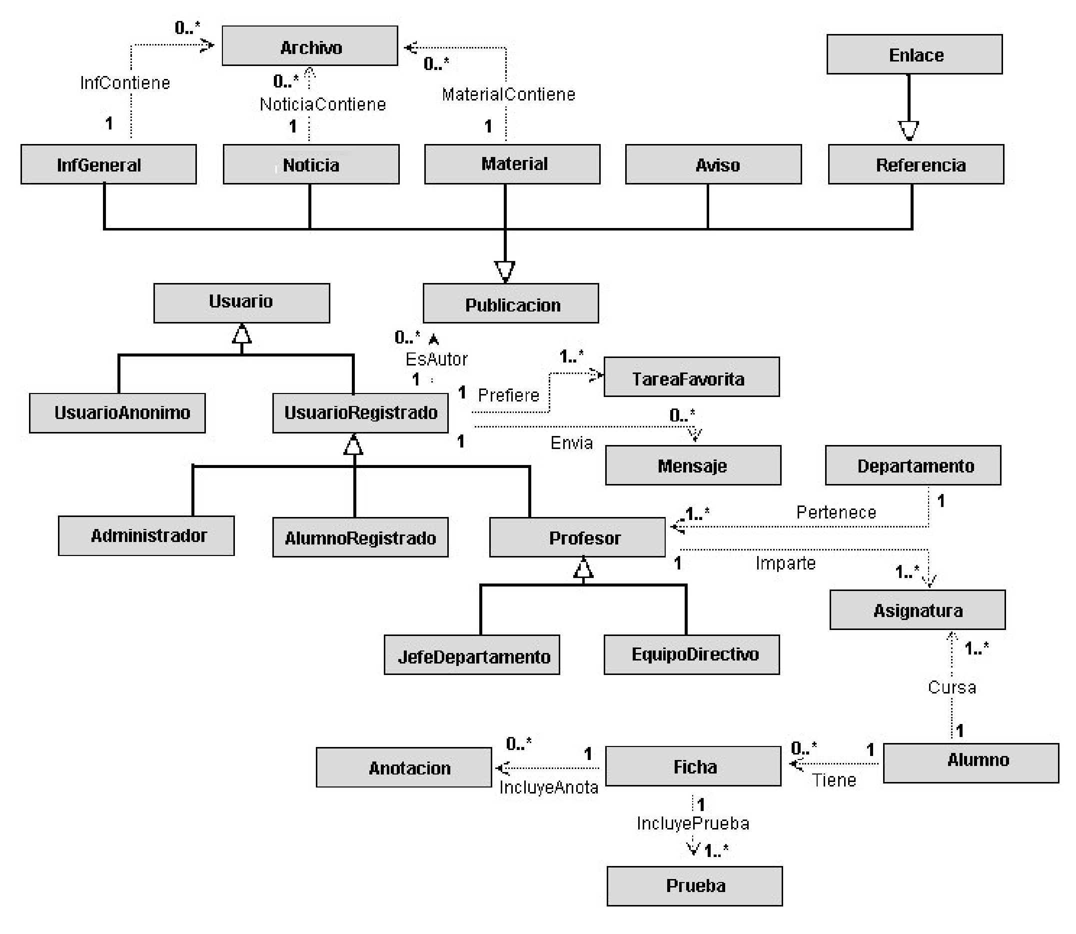


Figura 4.7. Diagrama de clases de GASPWIES.

4.2.- FASE DE DISEÑO.

La fase de diseño se ha dividido en los aspectos más importantes: el diagrama de clases y el diseño del sitio Web.

4.2.1.- Diagrama de clases.

Debido a la extensión del diagrama de clases, se ha simplificado en la Figura 4.7, incluyendo exclusivamente el nombre de las clases para posteriormente especificar los atributos y métodos.

NOMBRE DE CLASE	ATRIBUTOS	MÉTODOS
Publicación	ident, titulo, texto, fecha	publicar(), modificar(), eliminar()
InfGeneral		
Noticia	privada, prioridad, dirweb, masinfo	
Material		
Aviso		
Referencia	url	
Enlace	tematica	
Archivo	ident, titulo, tipoContenido, tamano	publicar(), eliminar()
Usuario	ident	
UsuarioRegistrado	password, nombre, apellidos, email	insertar(), registrar(), modificar(), eliminar(), cambiarRol(r:listaroles)

NOMBRE DE CLASE	ATRIBUTOS	MÉTODOS
UsuarioAnonimo		
Administrador		
AlumnoRegistrado		
Profesor	cargo, horaTutoria	
JefeDepartamento		
EquipoDirectivo		
TareaFavorita	idTarea, idEnlace	agregar(), eliminar()
Mensaje	idEmisor, idReceptor, texto, leido, fecha	enviar(), leer(), marcar(), eliminar()
Departamento	ident, titulo	anadir()
Asignatura	idAsignatura, titulo	anadir(), eliminar()
Alumno	idMatricula, password	consultarFicha()
Ficha	IdFicha	preparar(), eliminar(), modificar()
Anotación	idAnotacion	insertar(), eliminar()
Prueba	idPrueba, titulo	insertar(), eliminar(), modificar()

4.2.2.- Diseño del sitio Web.

Teniendo en cuenta el diagrama de requisitos, para realizar el diseño del sitio Web, se estudia:

a) *Página inicial*. Se decide dividirla en cuatro partes:

- La parte superior que incluirá logotipos, la fecha del sistema y enlaces para búsqueda de información y ayuda.
- Debajo habrá una franja para incluir las tareas favoritas de los usuarios registrados.
- El resto de la pantalla quedará dividido en dos partes. La parte izquierda estará dedicada a la navegación. Incluirá la bienvenida al usuario y un mensaje de aviso si tuviera mensajes nuevos.
- La parte derecha se dedicará a visualizar las páginas. Inicialmente, los datos del Centro con los titulares de las noticias novedosas con su fecha de publicación.
- En portada aparecerá una fotografía del Centro y sus datos de contacto.

b) *Navegación*. Decisiones tomadas:

- En la parte superior se dará la bienvenida al usuario.
- El menú de navegación se dividirá en dos partes: el contenido del sitio Web y una parte personal según el perfil del usuario registrado.
- Aquellas opciones que estén prohibidas al usuario por su perfil no se le mostrarán.
- Siempre existirá un enlace para validarse o desconectarse del portal.
- Siempre existirá un enlace en la parte superior de la navegación que facilite el acceso a la página de inicio.
- Se considerará como tarea favorita para todos los usuarios un enlace al mapa del sitio.
- En todas las páginas (en la parte derecha) aparecerá un camino con la ruta hasta llegar a la situación de la página con la posibilidad de navegar en sus antecesores y hasta la raíz.
- Las opciones que aparezcan en el menú de navegación del contenido del portal estarán determinadas por el rol del usuario conectado.
- La Figura 4.8 muestra la estructura del menú de la página de inicio en función de los distintos roles de usuario.
- La Figura 4.9 muestra la estructura completa del menú de navegación en el contenido para un usuario anónimo.

- La Figura 4.10 muestra los menús de navegación personales para los distintos tipos de rol de usuario.



Figura 4.8. Menú de inicio para los distintos roles.



Figura 4.9. Estructura completa de navegación pública del contenido.

c) *Interfaz de usuario.* Se toman las siguientes decisiones:

- Para conseguir la homogeneidad en las fuentes, colores y estilos, todas las páginas harán referencia a dos hojas de estilo (CSS): una para la navegación y otra para el contenido.
- Se incluirán tres juegos de interfaz para que el usuario elija el que más le guste. El usuario podrá añadir los que considere oportunos, tal como se refleja en el modelo de casos de uso.

- Habrá un juego de iconos e imágenes de botones para la navegación por si el usuario decide utilizar el formato gráfico. El usuario podrá añadir los que considere oportunos.
- Se podrá personalizar el número de noticias que se visualizan en el tablón de anuncios en cada página.

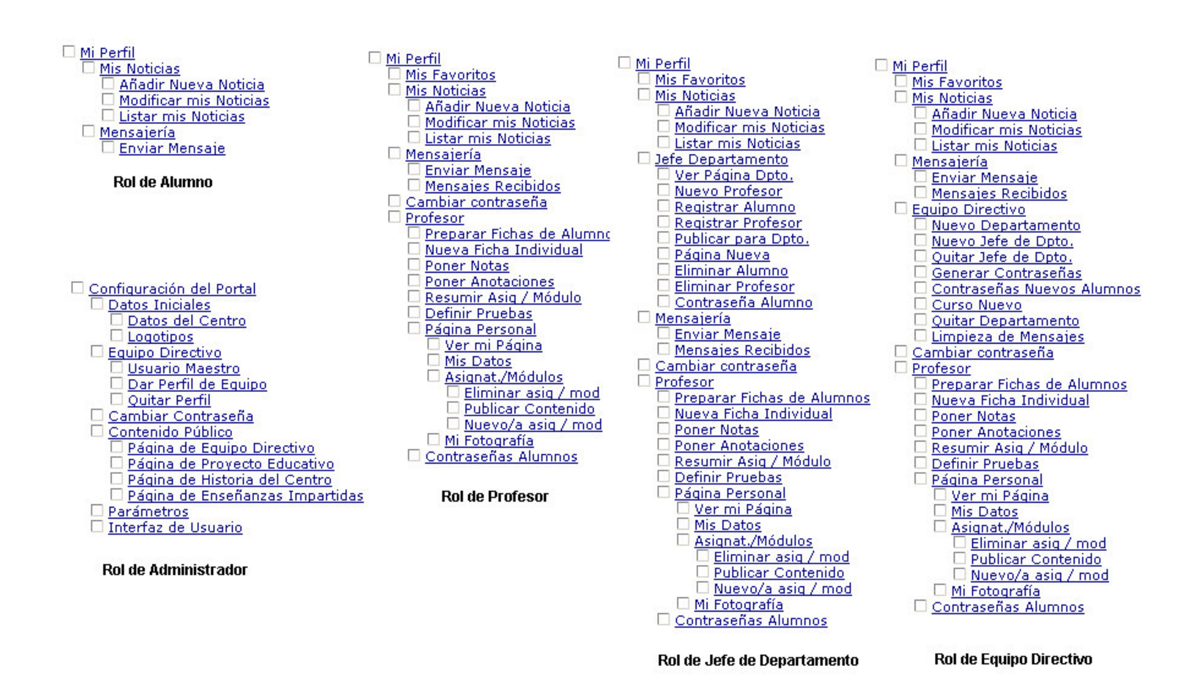


Figura 4.10. Estructura de navegación para los distintos roles.

4.3.- FASE DE IMPLEMENTACIÓN.

En los procesos de desarrollo de software la fase de implementación es una fase en la que fundamentalmente se traslada la fase de diseño al lenguaje de programación elegido. En este caso, la fase de implementación lleva asociadas determinadas particularidades que condicionan la fase. En primer lugar, es una aplicación cliente-servidor desarrollada para la Web y en segundo lugar, la aplicación se implementa sobre un servidor de aplicaciones como es Zope.

Las características que se consideran más importantes para destacar en esta fase son: la persistencia de la información, la estructura del sitio y la seguridad.

4.3.1.- Persistencia de la información.

Este punto es uno de los más delicados de la aplicación ya que, en principio, podría parecer razonable que el servidor de aplicaciones asumiera toda la responsabilidad de almacenamiento de la información. En cambio, existe una premisa obtenida en la fase de captura de requisitos que obliga a tomar una decisión alternativa. Como se expone en el apartado 4.1.2, los Centros almacenan la información de matrículas de alumnos, expedientes, profesores, etcétera en una base de datos Microsoft Access.

Este hecho, ha resultado determinante para decidir que parte de la persistencia se delegue en una base de datos Microsoft Access compatible en diseño con la base de datos de la aplicación IES2000. La justificación se basa en dos razones fundamentalmente: por un lado, se evitará que determinada información se tenga que introducir por segunda vez (datos de los alumnos, Departamentos del Centro y datos de los profesores), ya que podrían generar inconsistencias; y por otro lado, se facilitará el desarrollo de páginas complementarias o alternativas en lenguajes de *scripting* como PHP o ASP.



Figura 4.11. Tablas de la base de datos de GASPWIES.

La base de datos incluye la información de los aspectos "docentes". Se almacenan los datos personales de los usuarios registrados (profesores y alumnos), toda la

información de los grupos y sus alumnos, las calificaciones de las pruebas, las fichas, etc. Como excepción, también se almacenan los mensajes enviados entre usuarios para facilitar posibles consultas estadísticas o de otra índole que se puedan plantear y que se resolverían con SQL. La Figura 4.11 muestra el diseño de la base de datos Microsoft Access. Por el contrario, la información generada por la publicación de información en el portal (noticias, materiales de trabajo, avisos, etc.) se almacenan como objetos del sistema Zope (se almacenan en su ZODB) como se explica en el siguiente apartado.

4.3.2.- Estructura del sitio Web.

La estructura del sitio ha sido diseñada teniendo en cuenta las características del servidor de aplicaciones Zope. Las claves que se han tenido en cuenta para el diseño de esta estructura han sido las siguientes:

- Con objeto de facilitar el sistema de navegación y, sobre todo, facilitar la inclusión de nuevos elementos sin tener que modificar las páginas que presentaran los menús de contenido, se decide considerar cada enlace en sistema de navegación del Web como un objeto *carpeta*.
- Tras tomar la decisión en la fase de diseño que cada página del sitio estaría dividida en tres secciones (superior, izquierda y derecha), se implementa utilizando marcos (*frames*) que facilitan la presentación del contenido. En cambio, la utilización de marcos lleva asociado un problema: es posible que la tarea asociada a un enlace necesite la recarga de otro(s) marco(s) diferente(s) del destino del enlace. Para solucionarlo, y en aras de facilitar la reutilización, el mantenimiento y dar genericidad a los enlaces se ha tomado la decisión de que cada enlace en la navegación apunte a la página raíz y se le pasen dos parámetros: la página que se visualizará en el marco izquierdo y la que se visualizará en el marco derecho. Para conseguir esta reutilización y genericidad, a cada objeto susceptible de ser "navegado" se le incluye dos propiedades: `indice` y `contenido` que representan la página de navegación asociada y la página de contenido respectivamente. En realidad, no todos los incluyen. La orientación a objetos y la adquisición permiten que elementos cuyo destino sea

el mismo que el de su antecesor no los necesiten. Un ejemplo de URL para un enlace es:

`/demo/index_html?p1=Indice/Centro;p2=Indice/Centro/Profesorado`

- Pero, ¿cómo se sabe el nombre del portal (`demo`)?. Si en los enlaces se pone `demo` para el resto de portales no funcionará. Realmente, el objeto del sitio Web, cuando se crea incluye una propiedad llamada `portal` cuyo valor se corresponde con el identificador introducido por el creador y con el nombre asignado a la carpeta raíz del sitio Web. Así se obtiene el identificador del sitio Web dinámicamente.
- El sistema descrito en los puntos facilita la inclusión en las páginas de contenido dos de las características más importantes del sitio: la ruta de navegación hacia la página y la posibilidad de agregar la página a la lista de tareas favoritas del usuario.
- Con estas decisiones la lógica de la navegación se reduce a un único objeto Zope. Una plantilla ZPT incluye la lógica necesaria para conseguirlo. Ver anexo I.
- Para que sea posible la navegación en modo gráfico cada objeto susceptible de ser "navegado" contiene una propiedad llamada `icono` cuyo valor es el nombre del icono (un archivo situado en la carpeta `Skins` y precedido por la letra `i`). De esta forma la propia plantilla de navegación anterior "sabe" que icono tiene que asociar a cada enlace. El gráfico de la etiqueta es un archivo con el nombre del icono sin que le preceda la letra `i`.
- En las aplicaciones Web que utilizan lenguajes de *scripting* de lado servidor tradicionales los diseñadores suelen crear una carpeta donde incluyen todos los *scripts* que se usarán. En este caso, no resulta conveniente. Un carpeta es un objeto que representa una parte del diseño. Los objetos tienen propiedades (atributos) y métodos (operaciones). Los métodos (*scripts)* deben estar dentro de su objeto. Este es el motivo de que los `scripts python` y los `ZSQL Method` no estén incluidos en una única carpeta, sino situados estratégicamente en el lugar que les corresponde (su objeto propietario). Por ejemplo, existe un método `visualizar_publicación` genérico y luego existe uno para visualizar noticias, materiales, etc., cada uno situado en su objeto correspondiente.

- También ha resultado determinante la seguridad. La navegación privada se ha situado en un lugar jerárquicamente independiente de la pública. Así, con definir los permisos apropiados en la raíz de parte privada, queda garantizada la seguridad de acceso a esa parte. Ver apartado siguiente.

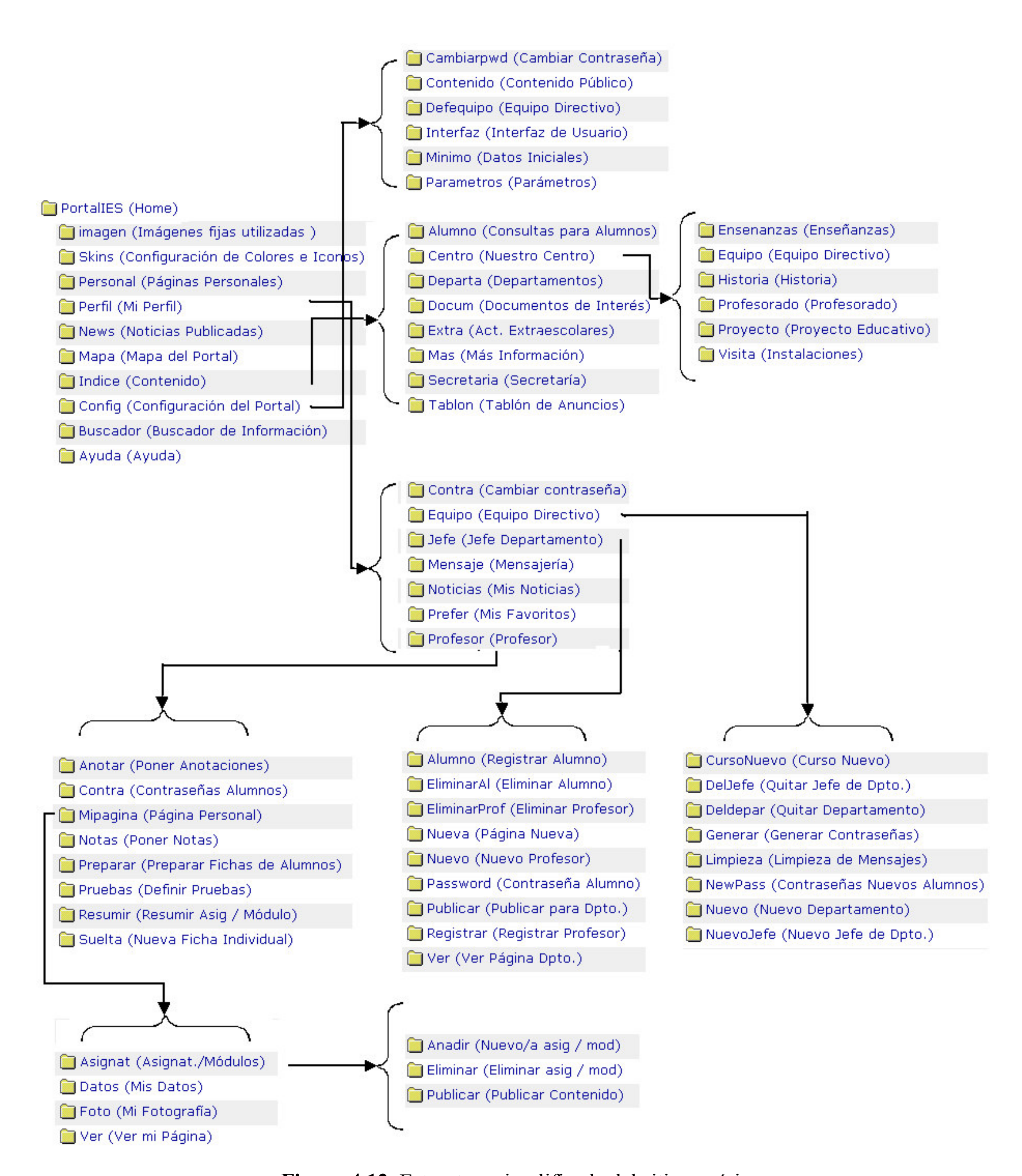


Figura 4.12. Estructura simplificada del sitio genérico.

- Los objetos del diagrama de clases que se decide que su persistencia se encuentre en el servidor Zope son, como se explicó, las publicaciones. Se decide que cada publicación se almacene en su contexto natural (por ejemplo, los materiales de trabajo publicado por un profesor, se almacena en el objeto correspondiente a su página personal). Las noticias se almacenan en una carpeta llamada `News` para facilitar su presentación.

La Figura 4.12 muestra la estructura del sitio (sin publicaciones, departamentos, páginas personales, etc).

4.3.3.- Seguridad.

La seguridad se ha implantado basándose íntegramente en las características del servidor de aplicaciones Zope. Ver apartado 3.6.3.5. Lo más destacable:

- Para evitar la autentificación a través de HTTP y realizarlo con aspecto de formulario y utilizando cookies, se ha utilizado un producto Zope llamado `CookieCumbler` (http://www.zope.org/Products).
- La lista de usuario registrados se almacena en un objeto *carpeta de usuario (acl_users)* situada en la raíz del portal.
- Inicialmente, existe el usuario `administrador` con contraseña `adm` con permisos para configurar el sitio. Este usuario tiene el rol `Manager` para administración del objeto sitio Web.
- El sistema Zope se encarga de la encriptación de contraseñas.
- Se han creado cuatro roles (perfiles) de usuario en la raíz del portal: `alumno, profesor, jefedpto` y `equipo.`
- Para completar la política de seguridad se han asignado los siguientes permisos (lo que no se especifica toma los permisos por defecto de Zope y para nombrar el objeto se supone que el sitio es `demo`):

OBJETO	PROPÓSITO	PERMISOS	
		DENEGAR ADQUISICIÓN	CONCEDER A
/demo/Config	Objeto que contiene las operaciones que permiten configurar el portal.	View	Manager[4]
/demo/Indice/Departa	Objeto que contiene los Departamentos Didáticos y su contenido.	Add Documents, Images and Files.	equipo, jefedpto
		Add Folders	equipo
		Delete Objects	equipo, jefedpto
		Manage Properties	equipo, jefedpto
/demo/Indice/Mas	Objeto que contiene los métodos para visualizar enlaces agrupados por temáticas, añadir temas y añadir enlaces.	Add Documents, Images and Files	alumno, profesor
		Add Folders	alumno, profesor
		Manage Properties	alumno, profesor
/demo/Indice/Centro/Visita	Objeto que contiene el método para visualizar las imágenes de las instalaciones del Centro, añadir imágenes y borrar imágenes.	Add Documents, Images and Files	equipo
		Delete Objects	profesor
/demo/News	Objeto que almacena las noticias publicadas y el método para visualizarlas.	Add Folders	Authenticated
		Manage Properties	Authenticated
		Add Documents, Images and Files	Authenticated
		Delete Objects	Authenticated
/demo/Personal	Objeto que almacena las páginas personales de los profesores.	Add Folders	Al propietario de la carpeta que contiene la pa´gina personal.
		Manage Properties	
		Add Documents, Images and Files	
		Delete Objects	
/demo/acl_users	Objeto que contiene la lista de usuarios del portal.	Manage Users	profesor
/demo/Perfil	Objeto que contiene las operaciones reservadas a los usuarios autentificados.	View	Authenticated

[4] En las siguientes concesiones no se incluye Manager, aunque como es lógico y razonable siempre se concede.

OBJETO	PROPÓSITO	PERMISOS	
		DENEGAR ADQUISICIÓN	CONCEDER A
/demo/Perfil/ Contra	Objeto con la operación de cambiar contraseña para los usuarios autentificados.	View	profesor
/demo/Perfil/ Prefer	Objeto con las operaciones de gestión de la lista de tareas favoritas	View	profesor
/demo/Perfil/ Mensaje	Objeto con las operaciones para gestión de mensajes instantáneos.	View	alumno, profesor
/demo/Perfil/ Mensaje/Recibo	Objeto con las operaciones para la gestión del buzón de mensajes recibidos.	View	profesor
/demo/Perfil/ Profesor	Objeto con las operaciones que pueden realizar los usuarios con perfil profesor.	View	profesor
/demo/Perfil/Jefe	Objeto con las operaciones que pueden realizar los usuarios con perfil Jefe de Departamento.	View	jefedpto
/demo/Perfil/ Equipo	Objeto con las operaciones que pueden realizar los usuarios con perfil Equipo Directivo.	View	equipo

5.- RESULTADOS.

El resultado de este proyecto es el desarrollo del sistema Generador Automático de un Sistema de Publicación Web para Institutos de Enseñanza Secundaria (GASPWIES). A continuación se explica cómo instalar la aplicación, se muestra un ejemplo de instancia del prototipo y, finalmente, se dan las pautas para realizar la utilización del manual de usuario.

5.1.- MÉTODOS DE INSTALACIÓN. REQUISITOS.

Para poder utilizar el GASPWIES se facilitan dos métodos de instalación en función del software que el usuario tenga instalado en su ordenador. Por un lado, cabe la posibilidad de que el usuario no tenga instalado el software exigido como requisito y también se contempla la posibilidad de que el usuario ya tenga instalado, al menos, un servidor Zope con versión igual o posterior a la requerida.

La aplicación es multiplataforma y se distribuye como un objeto Zope almacenado en un archivo (`PortalGenericoIES-1.0.tar.gz`) que es la forma estándar de distribución de objetos creados por los desarrolladores de Zope. No obstante, para la comodidad de instalación del usuario inexperto en Zope, se facilita un método de instalación completa que incluye todo el software necesario.

5.1.1.- Instalación completa.

Este método de instalación se recomienda a los usuarios que no tienen experiencia con Zope, a los que deseen una prueba rápida y "sin complicaciones" en la instalación y a los que tengan problemas al utilizar el método de instalación personalizada. Para la correcta instalación de la aplicación necesitan los siguientes requisitos técnicos mínimos:

- Pentium II 400 Mhz.
- 64 Mb de RAM.
- Microsoft Windows NT, 2000, 2003 o XP.
- Controlador ODBC para orígenes de datos de Microsoft Access (`.mdb`).

- Internet Explorer 4.0, Netscape Navigator 5 u otro navegador que soporte las funcionalidades de los anteriores.
- Disco duro con 35 Mbytes libres.

Los pasos para la instalación son los siguientes:

Paso 1.- Abrir la carpeta `Completa` del CD. Ejecutar el archivo `setup.exe` o `autosetup.exe`. Estos archivos han sido generados con el software de libre uso Inno Setup Compiler 4.2.7. El proceso está diseñado para aceptar todos los pasos que se presentan por defecto y finalmente pulsar el botón `Install` para proceder a la instalación. Este proceso de instalación (para ambos archivos) realiza las siguientes tareas:

- Instala la versión 6.2 del servidor Zope en la carpeta C:\GASPWIES.
- Incluye los productos `CookieCumbler 1.1+` y `ZODBCDA`.
- Crea un grupo de programas llamado *Generador Automático de un Sistema de Publicación Web para IES*, que contiene un acceso directo a la puesta en marcha del servidor y otro para su desinstalación.
- Prepara accesos directos para la puesta en marcha del servidor (C:\GASPWIES\start.bat) de forma opcional, según las decisiones tomadas en el proceso de instalación.
- Incluye en el servidor un portal para un Instituto de Enseñanza Secundaria (`demo`) para su configuración y explotación.

Si se ejecuta `autosetup.exe` además el proceso de instalación añade el DSN de usuario necesario para poder pasar al paso 3 directamente. Se necesitan privilegios para modificar el registro de Windows.

Paso 2.- Sólo para `setup.exe`. Crear un origen de datos que apunte la base de datos suministrada. Para ello, en el panel de control de Windows acceder al icono *Orígenes de Datos* (la forma de acceso y el nombre pueden variar en función de la versión de Windows) y agregar un DSN de sistema con la siguiente información:

- Controlador `Microsoft Access Driver`.
- Nombre del origen de datos `demo`.
- Seleccionar base de `datos C:\GASPWIES\datos.mdb`.

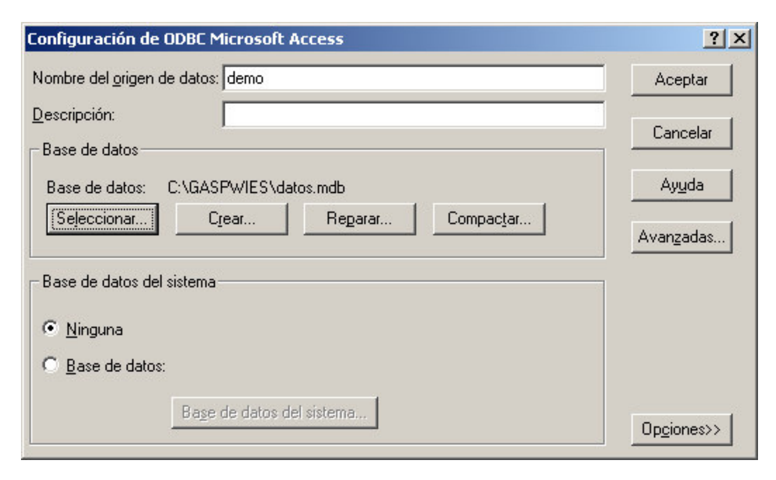


Figura 5.1. Agregar un origen de datos .mdb.

Paso 3.- Para probar el funcionamiento de la aplicación:

- Abrir un navegador Web.
- Teclear la URL http://localhost:8080/demo. Como se aprecia en la Figura 5.2 es necesario que inicialmente se configure el portal con una serie de información mínima para que el portal esté operativo.

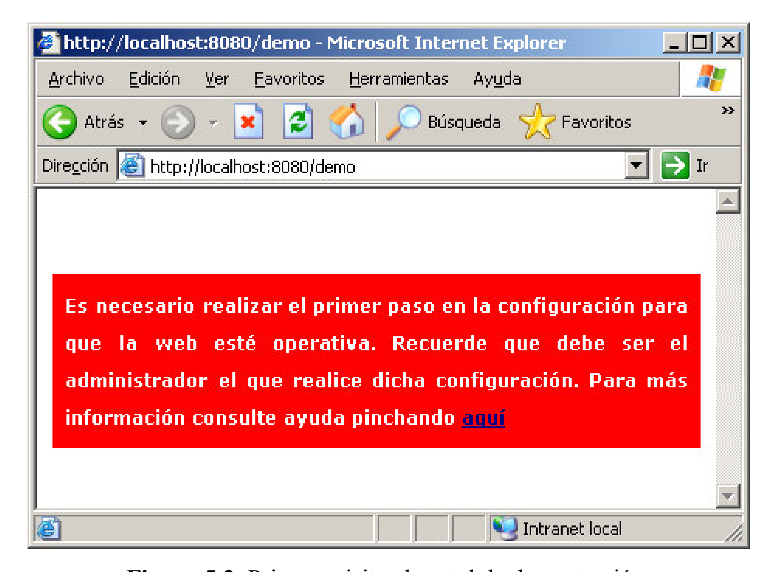


Figura 5.2. Primera visita al portal de demostración.

- Teclear la URL http://localhost:8080/demo/Config para configurar la información mínima (*Datos Iniciales)*. Para ello, se necesita validación con perfil Administrador: utilizar el usuario `administrador` y la contraseña `adm`.

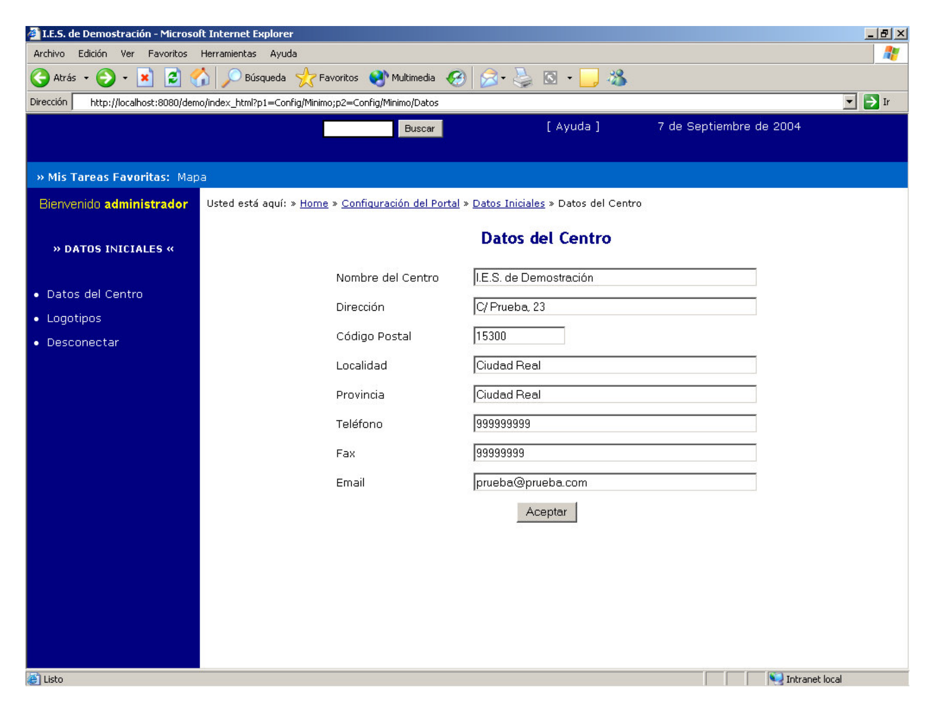


Figura 5.3. Datos mínimos para activación del portal.

- Una vez introducidos los datos del Centro necesarios para activar el portal, ya se puede visitar la página de inicio del portal recién creado, tecleando http://localhost:8080/demo o pinchando en el enlace `Home`.

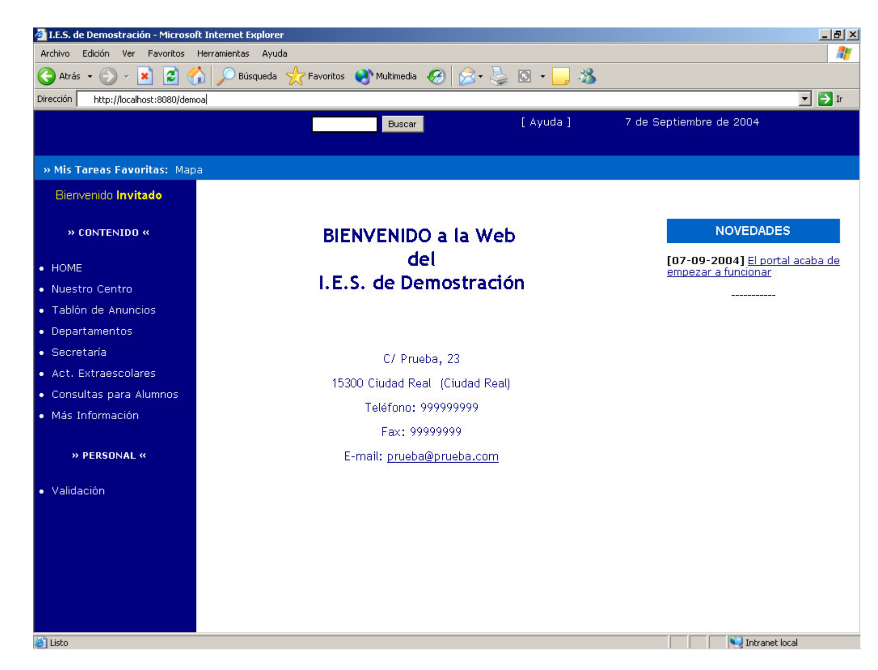


Figura 5.4. Primera visita al portal recién creado.

- Pasar a leer el manual de usuario para continuar la explotación del portal.

Paso 4.- Este paso es opcional. Si se desea administrar el portal como administrador de Zope, se tecleerá en el navegador http://localhost:8080/demo/manage y se introducirá la siguiente información de autentificación: usuario `admin` y contraseña `admin`.

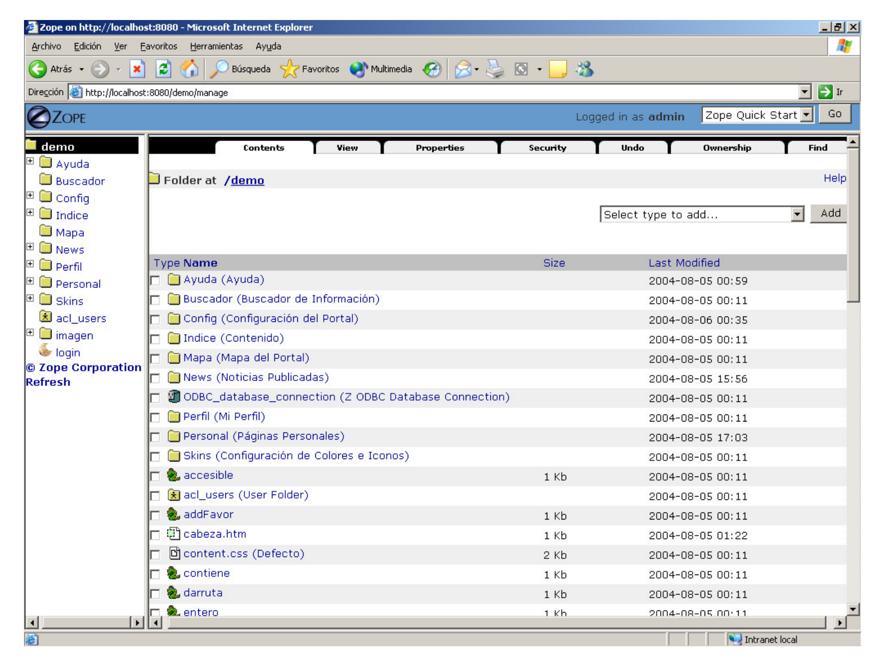


Figura 5.5. Administración Zope del portal `demo`.

Paso 5.- Este paso es opcional. Si se desea crear portales adicionales es necesario validarse como administrador de Zope utilizando el usuario `admin` y la contraseña `admin` en la URL http://localhost:8080/manage. A continuación se debe añadir un objeto `Portal Generico IES` en la lista de objetos que presenta Zope. No se debe olvidar crear una copia de la base de datos `C:\GASPWIES\datos.mdb` y crear un origen de datos con las características del creado para `demo` pero con el identificador que se haya dado al objeto Zope. Si se crea primero el objeto Zope y después el origen de datos, es necesario reiniciar el servidor Zope. Posteriormente, volver al paso 3 sustituyendo `demo` por el identificador que se haya dado al objeto Zope.

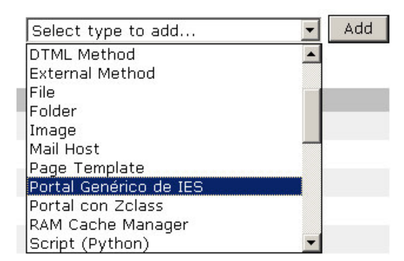


Figura 5.6. Añadir nuevos portales para IES.

5.1.2.- Instalación personalizada.

Este método de instalación necesita tener instalada la versión 2.6.0 o superior del servidor Zope. Los usuarios con experiencia en añadir productos al servidor Zope pueden utilizar este método de instalación. También es el método que deben utilizar los usuarios con sistema operativo UNIX-LINUX. Los únicos requisitos técnicos necesarios son los siguientes:

a) Controlador ODBC para orígenes de datos de Microsoft Access (`.mdb`).
b) Internet Explorer 4.0, Netscape Navigator 5 u otro navegador que soporte las funcionalidades de los anteriores.
c) Disco duro con 2 Mbytes libres.

Los pasos para la instalación son los siguientes:

Paso 1.- Abrir la carpeta `Personal` incluida en el CD. La carpeta contiene un archivo llamado `PortalGenericoIES.zip`. Este archivo, a su vez, incluye:

- `CookieCumbler1.1.tar.gz`, producto que permite la validación en el sistema mediante formularios utilizando cookies.
- `ZODBCDA-3.1.tgz`, producto que permite la utilización de ODBC con el sistema Zope. Este producto es estable con el lenguaje Python 2.1 y no funciona con las versiones 2.3.x del lenguaje. Si se dispone de la versión 2.7.0 de Zope o superior, debe sustituirse la instalación de Python 2.3 por la versión 2.1 o bien, visitar http://www.zope.org/Products y comprobar si existe una versión estable de ZODBC para Python 2.3.x (en el momento de la redacción de este documento existe una versión "beta").
- `PortalGenericoIES-1.0.tar.gz`, producto que contiene GASPWIES.
- `Datos.mdb`, base de datos creada con Microsoft Access 2000 que contiene las tablas que necesita la aplicación.
- `README.txt`, archivo que una breve descripción del producto, los requisitos y una breve explicación de la instalación. Este archivo se incluye por convenio en todos los productos desarrollados para el sistema Zope.

Paso 2.- Este paso sólo debe realizarse si no se tiene instalado ZODBCDA o posterior. Descomprimir el contenido del archivo `ZODBCDA-3.1.tgz` en la carpeta `lib/python/Products` del directorio raíz de su instalación Zope.

Paso 3.- Este paso sólo debe realizarse si no se tiene instalado CookieCumbler1.1 o posterior. Descomprimir el contenido del archivo `CookieCumbler1.1.tar.gz` en la carpeta `lib/python/products` del directorio raíz de su instalación Zope.

Paso 4.- Descomprimir el contenido del archivo `PortalGenericoIES-1.0.tar.gz` en la carpeta `lib/python` del directorio raíz de su instalación Zope. La carpeta `Products` contendrá tres nuevas carpetas: `CookieCumbler, ZODBCDA y PortalGenericoIES`.

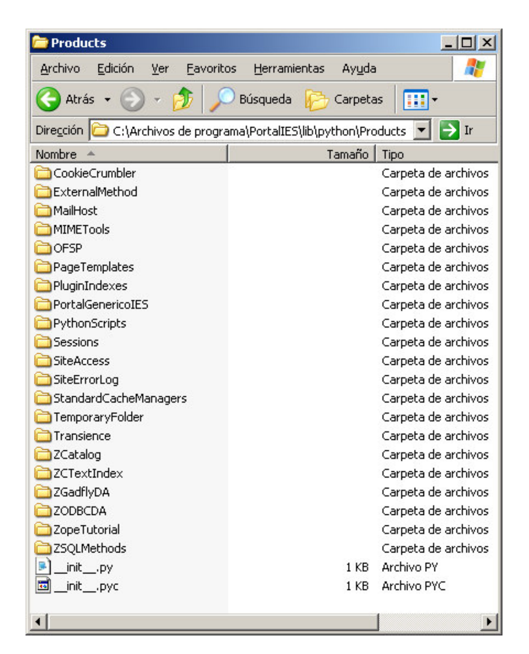


Figura 5.7. Contenido de la carpeta `Products`.

Paso 5.- Extraer el archivo `datos.mdb` del archivo .zip y realizar una copia por cada portal que se vaya a generar.

Paso 6.- Crear un origen de datos para cada portal que se vaya a generar, teniendo en cuenta que el nombre del origen de datos será el identificador del portal y que cada origen de datos debe apuntar a una copia diferente de `datos.mdb`. Ver el paso 2 del apartado correspondiente a la instalación completa.

Paso 7.- Iniciar el servidor Zope (reiniciar si ya estaba iniciado).

Paso 8.- Abrir una instancia del navegador de Internet y teclear la URL que permite administrar el servidor Zope, por ejemplo http://localhost:8080/manage y añadir un objeto `PortalGenericoIES` con el identificador que se haya dado al origen de datos. Ver Figura 5.6.

Paso 9.- Para ver el funcionamiento de la aplicación, ver el paso 3 del apartado correspondiente a la instalación completa, teniendo en cuenta que debe sustituirse el portal de ejemplo `demo` por el identificador elegido para el portal en el paso anterior.

5.1.3.- Flujo de trabajo para la explotación.

Una vez que el sitio Web está activado, existe un flujo de trabajo predeterminado para su explotación como muestra la Figura 5.8.

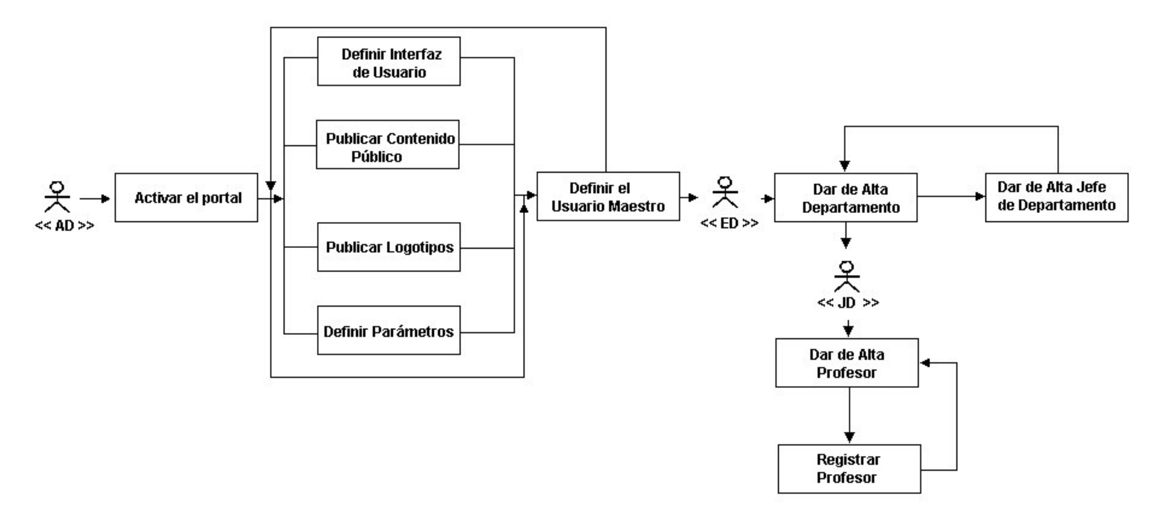


Figura 5.8. Flujo de trabajo para la explotación de GASPWIES.

Algunos pasos forman parte de la configuración (y es responsabilidad del usuario `administrador`) y otros de las actividades habituales del Centro.

El "usuario maestro" es el primer usuario creado con tareas distintas de la administración. Lo normal es que este usuario corresponda al Director del Centro o a un profesor al que éste le delegue la responsabilidad. Puede darse el caso de que este usuario

forme parte de un Departamento Didáctico incluido en la base de datos o que no sea así. En este segundo caso, es necesario dar de alta el Departamento para posteriormente registrar el "usuario maestro".

Usted está aquí: » Home » Configuración del Portal » Equipo Directivo » Usuario Maestro

Registrar Usuario como Equipo Directivo

Paso 1: Elección del Departamento al que pertenece este usuario

Lista de Departamentos

Seleccione Departamento
Actividades Extraescolares
Biología
Educación Física
Filosofía
Física y Química
Francés
Geografía
Griego
Inglés
Latín
Lengua
Matemáticas
Música
Orientación

Aceptar

Si la lista aparece vacía o el profesor pertenece a un departamento que NO aparece en la lista pulse AQUÍ

Usted está aquí: » Home » Configuración del Portal » Equipo Directivo » Usuario Maestro

Departamento Nuevo

Paso 1: Elección del Departamento al que pertenece este usuario

Nombre completo del Departamento (p.e. Informática)
Abreviatura (4 dígitos máximo)

Aceptar

Usted está aquí: » Home » Configuración del Portal » Equipo Directivo » Usuario Maestro

Registrar Usuario como Equipo Directivo

Paso 2: Introducir datos del profesor

*Los campos marcados con * son obligatorios*

* Nombre
* Apellidos
Cargo
Hora de Tutoría para padres
Correo electrónico
* Usuario
* Contraseña
* Repetir Contraseña

Alta y Registro de Usuario de Equipo Directivo

Figura 5.9. Registro del "usuario maestro".

En este punto, el usuario recién creado puede validarse y realizar el resto de tareas de la Figura 5.8 desde las opciones del perfil (Rol) Equipo Directivo. Es responsabilidad del "usuario maestro" crear los Departamentos Didácticos (incluido el suyo, porque al crear el usuario maestro se ha incluido en la base de datos pero no se ha insertado como objeto de la carpeta de Departamentos). El procedimiento para dar de alta un Departamento es similar al paso 1 de la creación del "usuario maestro". Se realiza en la ruta: » `Home » Mi Perfil » Equipo Directivo » Nuevo Departamento.`

Una vez que los Departamentos están dados de alta en el sitio Web, el "usuario maestro" debe dar de alta al profesor que ejerce las funciones de Jefe de Departamento. Se realiza en la ruta: » `Home » Mi Perfil » Equipo Directivo » Nuevo Jefe de Dpto` Pueden darse tres situaciones: que el profesor en cuestión sea nuevo en el Centro o no forme parte del portal (caso inicial), que ya haya sido de alta en el Departamento pero no esté registrado como usuario y que ya esté registrado. Tras elegir cuál de las situaciones es la apropiada se debe rellenar un formulario según el caso como se muestra en la Figura 5.10.

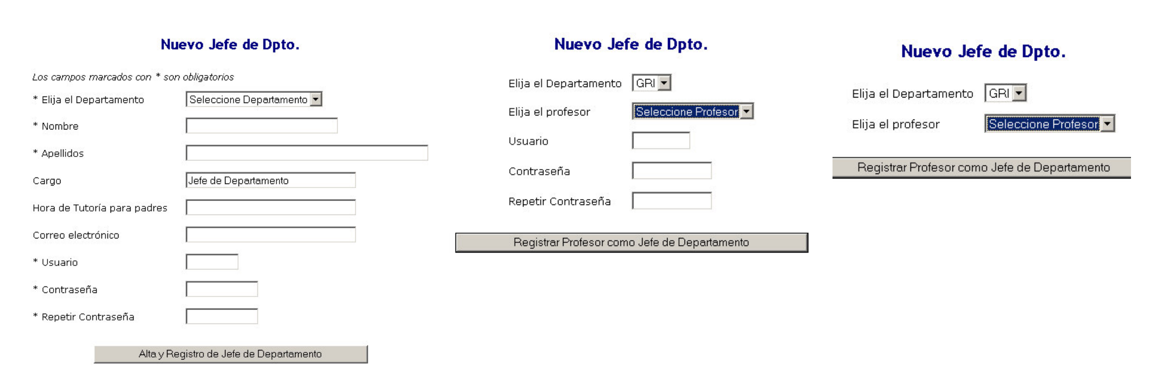


Figura 5.10.- Dar de alta un Jefe de Departamento.

En este punto, "termina" la responsabilidad del "usuario maestro" en el flujo de trabajo. Es el momento de que los Jefes de Departamento den de alta en la base de datos a los profesores miembros y registren a los profesores que lo deseen (sólo los usuarios registrados pueden publicar en el portal). Para hacerlo, primero se deben dar de alta los profesores en la ruta » `Home » Mi Perfil » Jefe Departamento » Nuevo Profesor` y posteriormente registrarlos en la ruta » `Home » Mi Perfil » Jefe Departamento » Registrar Profesor` como usuario rellenando los formularios de la Figura 5.11.

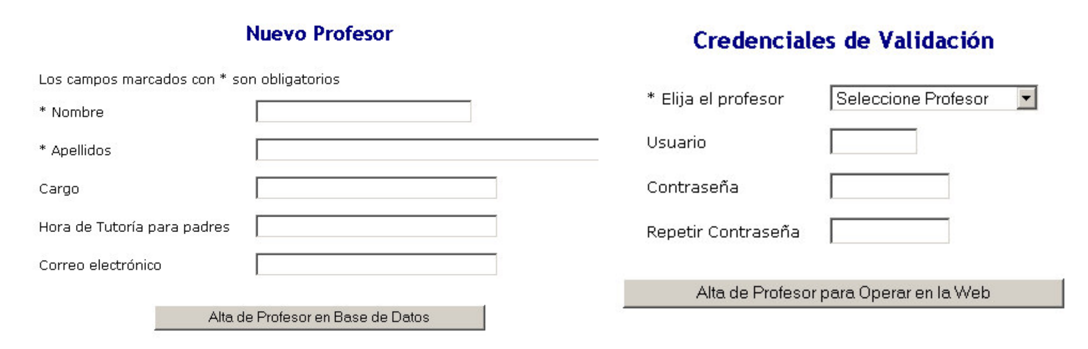


Figura 5.11. Dar de alta y registrar profesores de un Departamento.

Tras este proceso inicial, cada uno de los usuarios registrados pueden realizar las operaciones que deseen y para las que estén autorizados. Una tarea que no se debe olvidar es dar perfil de Equipo Directivo a los nuevos profesores que lo tengan y que acaban de ser registrados por su Jefe de Departamento.

5.2.- UN EJEMPLO DE INSTANCIA: I.E.S. ALJADA.

Durante el desarrollo de GASPWIES se ha estado utilizando un sitio Web para la realización de pruebas. El Instituto de Enseñanza Secundaria Aljada está situado en la pedanía murciana de Puente Tocinos. Puede decirse que se ha utilizado de "conejillo de indias" para la realización de pruebas a medida que avanzaba la implementación. La Figura 5.12 corresponde a la página inicial de su sitio Web. Es posible acceder a la página en la dirección http://www.iesaljada.com.



Figura 5.12. Página inicial de un sitio generado con GASPWIES .

También existe un portal de demostración accesible en todas sus operaciones (excepto cambiar la contraseña de administrador) en la dirección http://80.75.33.31:8080/demo. Para configurar el portal debe utilizarse el usuario `administrador` y la contraseña `adm`.

5.3.- MANUAL DE USUARIO.

En [POWELL 01] se apunta que el manual de usuario puede convertirse en un montón de papeles que nunca se sabe quién lo tiene y que al que el usuario le tiene cierta animadversión. Para este proyecto se ha decidido realizar el manual de usuario en formato legible por un navegador de Internet, dado que la aplicación es una aplicación Web.

El manual de usuario está integrado en la aplicación. Existe un enlace permanente en la parte superior de cada página de la aplicación. También puede accederse en la dirección http://localhost:8080/demo/Ayuda (suponiendo que `demo` es el identificador del sitio Web). La Figura 5.13 muestra una pantalla del contenido de la ayuda.



Figura 5.13. Ayuda en la aplicación GASPWIES.

Este manual de está diseñado e implementado siguiendo los principios de diseño de la aplicación, lo que beneficiará la facilidad de usabilidad y permitirá el acceso a todos los usuarios desde cualquier lugar.

Como puede apreciarse en la Figura 5.13, al igual que en la aplicación, existe una opción de búsqueda en el manual de usuario, un mapa de la ayuda, una lista de navegación jerárquica en cada página, etc.

6.- CONCLUSIONES Y PROPUESTAS.

6.1.- CONCLUSIONES.

El desarrollo de GASPWIES puede permitir la unificación de la estructura navegacional y el contenido de los sitios Web pertenecientes a los Institutos de Enseñanza Secundaria. Se ha conseguido eliminar el principal inconveniente que impide a los Centros tener su sitio en Internet: para realizar un sitio que ofrezca información actualizada y útil es necesario la colaboración de profesores expertos en la materia.

La aplicación está diseñada de forma que puede utilizarse en todos los Institutos de Enseñanza Secundaria, ya que se han tenido en cuenta todas las enseñanzas impartidas y todos los Departamentos Didácticos implicados.

GASPWIES ha aprovechado la existencia obligatoria en los Centros de la base de datos del IES2000[5] para que su base de datos sea compatible y poder así extraer la información más tediosa de facilitar a la aplicación, como es la relación de grupos con sus listas de alumnos.

Se ha intentado que los Centros puedan conseguir que su sitio Web no sea "como el de otro". Sus señas de identidad (historia, instalaciones, Proyecto Educativo, claustro de profesores, etc.) se pueden incluir con la mayor libertad en diseño y contenido. Los administradores pueden personalizar en gran medida el interfaz Web: logotipos, imágenes del centro, número de noticias a visualizar en portada, colorido, fuentes, iconos y gráficos de navegación, etc. Y todo esto sin necesidad de saber nada sobre diseño de páginas Web.

Se han incluido funcionalidades que pueden ayudar a los miembros de la Comunidad Educativa que son inexpertos en el uso de Internet y las nuevas tecnologías a formar parte activa del sitio Web:

---

[5] IES2000 es una aplicación que utilizan todos los Institutos de Enseñanza Secundaria para gestionar los expedientes de los alumnos, los profesores del Centro, etc. Almacena la información en una base de datos Microsoft Access.

- El manual de usuario está integrado en el portal (la distribución en papel nunca se sabe dónde está, quién la tiene y sobre todo, da mucha pereza leerlo) de manera que es una parte más de la navegación del sitio (estructura, diseño, interfaz y funcionalidad).
- A los usuarios registrados en el portal se les ha facilitado una lista de tareas favoritas que les permita el acceso rápido a las opciones deseadas.
- Cada página incluye su situación en el portal, facilitando al usuario conocer donde ésta y ofreciéndole la posibilidad de acceder directamente a los nodos de la ruta que va desde la raíz del portal a la página visitada.
- Todos los usuarios tienen la posibilidad en cada momento de visualizar un mapa del sitio Web que les permita acceso rápido a la opción deseada.
- Se ha incluido una opción de búsqueda en la estructura y contenido del portal, pensando en que una utilización intensa puede generar gran volumen de contenido.

GASPWIES es un software multiplataforma. Funciona sobre las versiones UNIX, Windows, Solaris y en todas en las que se distribuye el servidor de aplicaciones Zope. Para su uso y distribución no se requiere software comercial, ya que todo el software que le da soporte tiene licencia GPL. Puede ampliarse y modificarse por desarrolladores de Zope al facilitarse su código fuente.

Existe la posibilidad de ampliar el contenido del portal debido a la escalabilidad de la aplicación desarrollada. El sistema de navegación está diseñado para que aprovechando la orientación a objetos del servidor y su patrón de reutilización llamado *adquisición*, pueda añadirse contenido nuevo en cualquier parte de la jerarquía, sin más que crear una carpeta con sus propiedades para la opción deseada.

Dado que la aplicación está realizada sobre el servidor de aplicaciones Zope, la suplantación de roles, la realización de operaciones sin permiso y en general, todas las acciones que pongan en peligro la privacidad de información, la integridad de los datos y la seguridad del sistema, están controladas con el sistema de seguridad que ofrece Zope.

6.2.- PROPUESTAS.

Cuando se desarrolla una aplicación, siempre se corre el riesgo de que a mitad del desarrollo (en el mejor de los casos) o en durante su explotación, el cliente piense que le gustaría poder hacer algunas cosas más. En este caso, el cliente no es una excepción y considera que la aplicación podría incluir:

- Un calendario perpetuo que sirviera de agenda para los usuarios registrados.
- En las publicaciones para los Departamentos Didácticos y para las páginas personales, el autor de la publicación pudiera decidir si hacerla pública o privada (para miembros registrados, para profesores, para alumnos afectados, etc.).
- Dar protagonismo al Consejo Escolar, incluyéndolo en la estructura, registrando a sus miembros, permitirles realizar sus publicaciones, etc.

En lo que se refiere a cuestiones de diseño e implementación se realizan las siguientes propuestas:

- Podría modificarse el diseño para globalizar la aplicación de forma que pudiera utilizarse en otras etapas educativas (Educación Infantil, Educación Primaria y Universidad).
- Resultaría interesante estudiar las modificaciones a realizar (serían pocas) para que la capa de presentación fuera multidispositivo.
- Para completar la distribución, se estudiaría la trasformación del producto generado en un objeto ZClass que facilitara la actualización de versiones.
- Aprovechar los productos desarrollados por la comunidad para ampliar la funcionalidad del sistema. Por ejemplo, incluir un foro con fcForum, estudiar la posibilidad de integrar un módulo de cursos de enseñanza, etc.
- Podría resultar interesante su migración a Plone, siempre que antes se comprobara que el interfaz de Plone y sus características no "echan hacia atrás al cliente".

Ciudad Real, 16 de septiembre de 2004

Fdo.: Juan Carlos Parrilla Peláez

7.- REFERENCIAS.

- [ADITEL] *Introducción a Zope + Apache + BD´s relacionales.* http://www.aditel.org/jornadas/02/ponencias/zope
- [BERSTEIN 02] Berstein, R. M.; Robertson, S. *Zope Bible.* Hungry Minds, 2002.
- [BMODEL] *The Booch Model*. http://www.ifra.ing.tu-bs.de/docs/BoochReferenz/
- [BOIKO 02] Boiko, B. *Content Management Bible.* Wiley Publishing, Inc. 2002.
- [BOOCH 01] Booch, G.; Rumbaugh, J. Y Jacobson, I. *El Lenguaje Unificado de Modelado.* Addison-Wesley, 2001.
- [BRUECK 01] Brueck, D.; Tanner, S. *Python 2.1 Bible.* Hungry Minds, 2002.
- [CACHERO 03] *Una extensión a los Métodos OO para el modelado y generación automática de interfaces hipermediales.* http://www.dlsi.ua.es/~ccachero/pTesis.htm
- [CMF] *CMF Project Web Site*. http://cmf.zope.org
- [COLADO 03] Colado, C. *Diseño y desarrollo de aplicaciones Web multidispositivo*. Germinus XXI, 2003.
- [CONALLEN 02] Conallen, J. *Building Web Applications with UML Second Edition.* Addison-Wesley, 2002.
- [COUTIN 01] Coutin, A. *Arquitectura de información para sitios Web*. Anaya Multimedia, 2001.
- [DCAH] *Diseño conceptual de aplicaciones hipermedia.* http://dei.inf.uc3m.es/espanyiol/miembros/paloma/upv/contenidos.htm
- [ENGLISH 03] English, B. *Microsoft Content Management Server 2002: A Complete Guide*. Addison-Wesley, 2003.

- [ESCALONA 02a] Escalona, M. J. y otros. *Desarrollo de la navegación en entornos Web.* Universidad de Sevilla, 2002.
- [ESCALONA 02b] Escalona, M. J.; Koch, N. *Ingeniería de Requisitos en Aplicaciones para la Web – Un Estudio Comparativo.* Universidad de Sevilla, 2002.
- [FEILER 99] Feiler, J., *Database-Driven Web Sites.* Morgan Kaufmann Publishers, San Francisco, California, 1999.
- [GALLI 01] Galli, R. *Desarrollo Web extremo.* Mallorca, 2001.
- [GIL 03] Gil, A. *Zope Page Templates.* Aditel, 2003.
- [GOMEZ 03] Gómez, M. P. *Herramientas Avanzadas para la Producción de Interfaces de Usuario Basados en Tecnologías Web.* UCLM, 2003.
- [GONZALEZ 02] González, C.; Cuevas, V. J. *Base de datos y Web.* UCLM, 2002.
- [HISPAZOPE] HispaZope. http://www.hispazope.org
- [ICTI] *ICTI Consulting: Características de Typo3.* http://www.icticonsulting.com/Typo3.44.0.html
- [JACOBSON 00] Jacobson, I.; Booch, G. y Rumbaugh, J. *El Proceso Unificado de Desarrollo de Software.* Addison-Wesley, 2000.
- [JDBC] *JDBC API tutorial and reference - Second Edition,* http://Java.sun.com/products/jdbc/, 2001
- [JSP] *Java Server Pages: Dinamically Generated Web Content.* http://www.Javasoft.com/products/jsp/
- [KOCH 00] Koch, N. *Comparing Development Methods for Web Applications.* Ludwig-Maximilians-University of Munich, 2000.
- [LATTEIER 00] Latteier, A; Pelletier, M. *Zope Book.* New Riders Publishing, 2000.
- [LLOYD 02] Lloyd, B. *An introduction to Zope.* Developer Shed, 2002.

- [LORES 02] Lorés, J.; Granollers, T. *La Ingeniería de la usabilidad aplicada al diseño y desarrollo de sitios Web*. Universidad de Lleida, 2002.
- [MCKAY] McKay, A. *Plone Book*, http://plone.org/documentation/book/.
- [MCMS02] *Microsoft Content Management Server 2002 Evaluation Guide.* http://www.hands-on-labs.com/only4gurus/techlib/microsoft/mcms_evalguid.doc
- [MIGUEL 01] de Miguel, T. *El sistema Zope.* Universidad Politécnica de Madrid, 2001.
- [NIELSEN 00] Nielsen, J. *Usabilidad. Diseño de sitios Web.* Prentice Hall, 2000.
- [OOSE] *Object Oriented Software Engineering. Caso Práctico.* http://www.itba.edu.ar/capis/rtis/articulosdeloscuadernosetapaprevia/petronio8.pdf.
- [PIATTINI 96] Piattini, M.G. y otros. *Análisis y diseño detallado de aplicaciones de gestión*. RA-MA, 1996.
- [POLO 04] Polo, M. *Apuntes de Ingeniería del Software II*. UCLM, 2004.
- [POWELLL 01] Powell, T. A. *Diseño de sitios Web. Manual de Referencia.* McGRAW-HILL, 2001.
- [PRESSMAN 98] Pressman, R. S. *Ingeniería del Software. Un enfoque práctico. 4ª Edición.* McGraw-Hill, 1998.
- [PRIETO 02] Prieto, O. J. *Introducción a las aplicaciones Web*. 2002.
- [ROBLES 03] Robles G.; González, J. M. *Introducción a Zope y Plone.* Universidad Rey Juan Carlos, 2003.
- [ROSENFELD 98] Rosenfeld, L.; Morville, P. *Information Architecture for the World Wide Web*, O'Reilly, 1998.
- [SERVLETS] *JavaTM Servlet Technology: The Power Behind the Server.* http://Java.sun.com/products/servlet

- [SILVA 02] Silva, D. A., Mercerat, B. *Construyendo Aplicaciones Web con una Metodología de Diseño Orientada a Objetos.* 2002.
- [SIMO 03] Simó, F.J. *Portal Colaborativo con Zope+Plone.* ETSI de Telecomunicaciones de Madrid, 2003.
- [TYPO3] *Typo3.* http://typo3.org, http://www.typo3.com
- [UWA] *UWA: Ubiquitous Web Applications.* http://www.uwaproject.org
- [VALDERAS] Valderas, P. J. *Análisis, Diseño e Implementación de un Portal Web para un Departamento Universitario.* http://bdpfc.inf.upv.es/theses/available/etd-07222002-133409/unrestricted/PFCValderas2002.pdf
- [XLS] *Extensible Stylesheet Language (XSL)*. http://www.w3.org/TR/xsl
- [XML] *Extensible Markup Language (XML). The base specifications.* http://www.w3.org/XML
- [ZOPE] *Zope Project Web Site*. http://www.zope.org

ANEXO I.- CÓDIGO FUENTE.

En este apartado se relaciona el código fuente correspondiente a las tareas más interesantes y didácticas desarrolladas para GASPWIES.

1. Documento DTML (`/demo/index_html`) que visualiza la página inicial del sitio Web.

```
<html>
<head>
<title><dtml-var ies></title>
<link rel="STYLESHEET" type="text/css" href="content.css">
</head>
<dtml-if "AUTHENTICATED_USER.has_role(['Manager'])">
<frameset framespacing="0" border="0" rows="55,30,*" frameborder="0">
   <frame name="cabeza" scrolling="no" noresize src="cabeza.htm">
   <frame name="favor" scrolling="no" noresize src="favoritos">
   <frameset cols="200,*">
    <frame name="indice" src="<dtml-var p1 missing="Indice">" scrolling="auto" noresize
                 marginheight="5">
    <frame name="contenido"  src="<dtml-var p2 missing="portada">" scrolling="auto"
                 noresize marginwidth="5" marginheight="5">
  </frameset>

<dtml-elif "visible=='S'">
<frameset framespacing="0" border="0" rows="55,30,*" frameborder="0">
   <frame name="cabeza" scrolling="no" noresize src="cabeza.htm">
   <frame name="favor" scrolling="no" noresize src="favoritos">
   <frameset cols="200,*">
    <frame name="indice" src="<dtml-var p1 missing="Indice">" scrolling="auto" noresize
        marginheight="5">
    <frame name="contenido"  src="<dtml-var p2 missing="portada">" scrolling="auto"
           noresize marginwidth="5" marginheight="5">
  </frameset>
</dtml-if>
<dtml-if "visible=='N'">
<br><br>
<p class="error">
 Es necesario realizar el primer paso en la configuración para que la web esté operativa.
Recuerde que debe ser el administrador el que realice dicha configuración.
 Para más información consulte ayuda pinchando <a href="Ayuda">aquí</a>
</p>

</dtml-if>


  <noframes>
  <body>
  <p>Esta página usa marcos, pero su explorador no los admite.</p>

  </body>
  </noframes>
</frameset>

</html>
```

2. Plantilla Zope (`/demo/favoritos`) que visualiza el marco con las tareas favoritas. Se incluye el código del *Script* Python (`/demo/getFavor`) que retorna una lista que contiene los objetos asociados a las tareas favoritas del usuario registrado.

```
<html>
  <head>
    <title tal:content="template/title">The title</title>
 <link rel="STYLESHEET" type="text/css" href="indice.css">

  </head>
  <body class="favor">
   <b>» Mis Tareas Favoritas:</b> 
   <img tal:condition="python:here.Skins.buena!='Texto'" width="10" height="10"
       tal:attributes="src python:'Skins/%s/imapa.gif'%here.Skins.buena">
   <a target="_top" tal:attributes="href python:'index_html?p1=Indice&p2=Mapa'">Mapa</a>
   <span tal:condition="python:here.getRol('profesor')=='true'">|
    <span tal:content="structure container/getFavor">ruta de naavegacion</span>

   </span>
  </body>
</html>
```

/demo/getFavor

```
from Products.PythonScripts.standard import html_quote

links=[]
lugar = context.restrictedTraverse('/%s/Skins' % (context.portal))
iconos = lugar.buena
if iconos!='Texto':
 lugar = context.restrictedTraverse('/%s/Personal/%s' %
     (context.portal,context.getUsuario()))
 if lugar.f5!='':
           enlace = context.restrictedTraverse(lugar.p52)
           links.insert(0, """<img src="/%s/Skins/%s/%s.gif">""" % \
           (context.portal,iconos,'i'+enlace.icono)
           +" "
            """<a href="/%s/index_html?p1=%s;p2=%s" target="_top">%s</a>""" % \
           (context.portal,html_quote(lugar.p51),html_quote(lugar.p52),
              html_quote(lugar.f5)))

 if lugar.f4!='':
           enlace = context.restrictedTraverse(lugar.p42)
           links.insert(0, """<img src="/%s/Skins/%s/%s.gif">""" % \
           (context.portal,iconos,'i'+enlace.icono)
           +" "
            """<a href="/%s/index_html?p1=%s;p2=%s" target="_top">%s</a>""" % \
           (context.portal,html_quote(lugar.p41),html_quote(lugar.p42),
             html_quote(lugar.f4)))

 if lugar.f3!='':
           enlace = context.restrictedTraverse(lugar.p32)
           links.insert(0, """<img src="/%s/Skins/%s/%s.gif">""" % \
           (context.portal,iconos,'i'+enlace.icono)
           +" "
            """<a href="/%s/index_html?p1=%s;p2=%s" target="_top">%s</a>""" % \
           (context.portal,html_quote(lugar.p31),html_quote(lugar.p32),
              html_quote(lugar.f3)))

 if lugar.f2!='':
           enlace = context.restrictedTraverse(lugar.p22)
           links.insert(0, """<img src="/%s/Skins/%s/%s.gif">""" % \
           (context.portal,iconos,'i'+enlace.icono)
           +" "
            """<a href="/%s/index_html?p1=%s;p2=%s" target="_top">%s</a>""" % \
           (context.portal,html_quote(lugar.p21),html_quote(lugar.p22),
               html_quote(lugar.f2)))

 if lugar.f1!='':
           enlace = context.restrictedTraverse(lugar.p12)
```

```
            links.insert(0, """<img src="/%s/Skins/%s/%s.gif">""" % \
            (context.portal,iconos,'i'+enlace.icono)
            +" "
             """<a href="/%s/index_html?p1=%s;p2=%s" target="_top">%s</a>""" % \
            (context.portal,html_quote(lugar.p11),html_quote(lugar.p12),
                  html_quote(lugar.f1)))

 return " | " . join(links)

else:

 lugar = context.restrictedTraverse('/%s/Personal/%s' %
        (context.portal,context.getUsuario()))
 if lugar.f5!='':
         links.insert(0, """<a href="/%s/index_html?p1=%s;p2=%s" target="_top">%s</a>""" %
   \ (context.portal,html_quote(lugar.p51),html_quote(lugar.p52), html_quote(lugar.f5)))

 if lugar.f4!='':
         links.insert(0, """<a href="/%s/index_html?p1=%s;p2=%s" target="_top">%s</a>""" %
   \       (context.portal,html_quote(lugar.p41),html_quote(lugar.p42),
            html_quote(lugar.f4)))
 if lugar.f3!='':
         links.insert(0, """<a href="/%s/index_html?p1=%s;p2=%s" target="_top">%s</a>""" %
            \
        (context.portal,html_quote(lugar.p31),html_quote(lugar.p32), tml_quote(lugar.f3)))

 if lugar.f2!='':
         links.insert(0, """<a href="/%s/index_html?p1=%s;p2=%s" target="_top">%s</a>""" %
        \(context.portal,html_quote(lugar.p21),html_quote(lugar.p22), tml_quote(lugar.f2)))

 if lugar.f1!='':
         links.insert(0, """<a href="/%s/index_html?p1=%s;p2=%s" target="_top">%s</a>""" %
   \    (context.portal,html_quote(lugar.p11),html_quote(lugar.p12), html_quote(lugar.f1)))

 return " | " . join(links)
```

3. *Script* Python (`/demo/accesible`) que retorna una lista de objetos contenedores a los que el usuario registrado tiene permiso de navegación.

```
# parámetros: here (contenedor) y user (usuario registrado)
request = context.REQUEST
RESPONSE =  request.RESPONSE

lista=[]
for folder in here.objectValues('Folder'):
   if user.has_permission('View',folder):
          lista.insert(5000,folder)

args=(('orden',),)


return sequence.sort(lista, args)
```

4. Plantilla Zope (`/demo/Perfil/index_html`) que permite la navegación sistemática y genérica de la parte privada de los usuarios registrados.

```
<html>
  <head>
    <title tal:content="template/title">The title</title>
 <link rel="STYLESHEET" type="text/css" href="indice.css">

  </head>
  <body>


<table class="normal" cellpadding="5" cellspacing="0" border="0" width="100%">
<tr>
<td colspan="2" class="bienve" align="center">
 Bienvenido <b tal:define="global usuario python:here.getUsuario()"
                  tal:condition="python:usuario != 'Anonymous User'"
                  tal:content="usuario"> user</b>
            <b><span tal:condition="python:usuario== 'Anonymous User'"> Invitado </span>
            </b>
</td>
</tr>

<span        tal:condition="python:usuario    !=    'Anonymous    User'"    tal:repeat="item
python:here.nuevoMail(destino=usuario)">
<tr tal:define="texto string:${item/texto}"
   tal:condition="python:texto>'0'">
<td class="mail">¡ Hay mensajes nuevos ! </td>
<td><a            href="direccion"            target="_top"            tal:attributes="href
python:here.getDestino('/%s/Perfil/Mensaje'%here.portal,'/%s/Perfil/Mensaje/Recibo'%here.po
rtal)" >
<img border="0" src="imagen/sms.gif" alt="Leer mensajes" height="20" width="20">
 </a>
</td>
</tr>
</span>
</table>
<p align="center" class="pequena">
  <b>» <span tal:content="python:here.toUpper(here.title)">CONTENIDO</span> «</b>
</p>

<span tal:condition="python:here.Skins.buena=='Texto'">

<table class="normal" cellpadding="5" border="0" width="100%">
<tr valign="top">
     <td><li>
       <a href="index_html" target="_top"
           tal:attributes="href python:here.getDestino('Indice','portada')">
        HOME</a></li>
     </td>
  </tr>

   <tr tal:define="subfolders python:here.accesible(here,user)"
      tal:repeat="folder subfolders"
      tal:condition="subfolders"
      valign="top">
 <td>      <li> <a href="folder" target="_top"
           tal:define="indice folder/indice;contenido folder/contenido"
           tal:attributes="href python:here.getDestino(indice,contenido)"
           tal:content="folder/title">folder id
        </a></li>
</td>
  </tr>
<tr>
      <td     class="normal"><li><a     target="_top"     href="www"     tal:attributes="href
python:'/%s/login/logout' % (here.portal)">
       Desconectar</a></li></td>
    </tr>

</table>
</span>

<span tal:condition="python:here.Skins.buena!='Texto'">
<table class="normal" cellpadding="5" cellspacing="0" border="0" width="100%">
```

```
<tr valign="center">
     <td>
       <img border="0" src="Skins/Defecto/home" width="10" height="10"
               tal:attributes="src python:'Skins/%s/ihome.gif' % here.Skins.buena">
      </td>
     <td>
       <a href="index_html" target="_top" tal:attributes="href
           python:here.getDestino('Indice','portada')">
        <img border="0" src="Skins/Defecto/home" alt="Home" width="170" height="18"
         tal:attributes="src python:'Skins/%s/home.gif' % here.Skins.buena;
                         onMouseOver python:here.getMouseOver('home_on.gif');
                         onMouseOut python:here.getMouseOver('home.gif')">

       </a>
     </td>
  </tr>
</table>

<table class="normal" cellpadding="5" cellspacing="0" border="0" width="100%" >
    <tr valign="center" tal:define="subfolders python:here.accesible(here,user)"
      tal:repeat="folder subfolders"
      tal:condition="subfolders">
     <td>
       <img border="0" src="Skins/Defecto/home" width="10" height="10"
         tal:attributes="src python:'Skins/%s/i%s.gif'%(here.Skins.buena,folder.icono)">
     </td>
     <td>
       <a href="folder" target="_top"
           tal:define="indice folder/indice;contenido folder/contenido"
           tal:attributes="href python:here.getDestino(indice,contenido)">
        <img border="0" src="Skins/Defecto/home" alt="Home" width="170" height="18"
         tal:attributes="alt folder/title;
                 src python:'Skins/%s/%s.gif'%(here.Skins.buena,folder.icono);
                         onMouseOver python:here.getMouseOver(folder.icono+'_on.gif');
                         onMouseOut python:here.getMouseOver(folder.icono+'.gif')">

       </a>
     </td>
   </tr>

    <tr>
     <td>
      <img border="0" src="Skins/Defecto/idesc" width="10" height="10"
       tal:attributes="src python:'Skins/%s/idesc.gif' % here.Skins.buena">
     </td>
      <td      class="normal"><a      target="_top"      href="www"      tal:attributes="href
python:'/%s/login/logout' % (here.portal)">
       <img border="0" src="Skins/Defecto/desc" alt="Desconectar" width="170" height="18"
        tal:attributes="src python:'Skins/%s/desc.gif' % here.Skins.buena;
                         onMouseOver python:here.getMouseOver('desc_on.gif');
                         onMouseOut python:here.getMouseOver('desc.gif')">

</a></td>
    </tr>

</table>

</span>
</body>
</html>
```

5. Script Python (/demo/darruta) que visualiza el cada página su situación como ruta de navegación.

```
"""
Da una cadena desde el directorio raíz (indice) con hiperenlaces
"""
from Products.PythonScripts.standard import html_quote

links=[]
links.insert(0, "%s" % \
                 (html_quote(context.title_or_id())))
for parent in context.REQUEST.PARENTS[1:]:
         links.insert(0, """<a href="/%s/index_html?p1=%s;p2=%s" target="_top">%s</a>""" %
          \ (context.portal,html_quote(parent.indice),html_quote(parent.contenido),
              html_quote(parent.title_or_id())))
         if parent.getId()==context.portal:
           links.insert(0, "Usted está aquí: ")
           break
return " » ".join(links)
```

6. Script Python (`/demo/getMisAsig`) que retorna una lista que contiene los objetos asociados a las asignaturas impartidas por el usuario registrado.

```
request = context.REQUEST
lista=[]
lugar = context.restrictedTraverse('/%s/Personal/%s' % (context.portal,quien))
for asig in lugar.objectValues('Folder'):
 lista.insert(0,asig)
return lista
```

7. Plantilla Zope (`/demo/Mapa/index_html`) que visualiza el mapa del portal en función del usuario registrado. Se incluye el código de los *Script* Python invocados.

```
<html>
  <head>
    <title tal:content="template/title">The title</title>
<link rel="STYLESHEET" type="text/css" href="content.css">
  </head>
  <body>
   <span  class="ruta" tal:content="structure container/darruta">ruta de naavegacion</span>
  <h1 tal:content="template/title"></h1>
    <table  border="0" class="titulotabla" cellpadding="3" width="100%">
    <tr valign="top">
     <td width="30%" align="center">Contenido del Portal
      </td>
      <td width="30%" align="center">Páginas Personales
      </td>
      <td width="40%" align="center"
           tal:condition="python:here.getRol('Authenticated')=='true'">
       Perfil de Usuario
      </td>
     </tr>
</table>
    <table  border="1" class="ruta" cellpadding="3" width="100%">
    <tr valign="top">
     <td width="30%"><span tal:repeat="item python:here.busca('Indice')">
           <span tal:content="structure item">
           </span>
         </span>
      </td>
      <td width="30%"><span tal:repeat="item python:here.busca('Personal')">
           <span tal:content="structure item">
           </span>
         </span>
      </td>
      <td   width="40%"   tal:condition="python:here.getRol('Authenticated')=='true'"><span
tal:repeat="item python:here.busca('Perfil')">
           <span tal:content="structure item">
           </span>
         </span>
      </td>
      <td   width="40%"   tal:condition="python:here.getRol('administrador')=='true'"><span
tal:repeat="item python:here.busca('Config')">
           <span tal:content="structure item">
           </span>
         </span>
      </td>
     </tr>
</table>

  </body>
</html>
```

/demo/Mapa/busca

```
#parámetros: donde (lugar de dónde debe hacerse el mapa)
request = context.REQUEST
from Products.PythonScripts.standard import html_quote
lista=[]

lugar = context.restrictedTraverse('/%s/%s' % (context.portal,donde))
lista.insert(9999, """<img src="/%s/Skins/%s/i%s.gif" width="10" height="10">""" % \
           (context.portal,context.Skins.buena,lugar.icono)
            +
            """ <a href="/%s/index_html?p1=Indice;p2=%s" target="_top">%s</a>""" % \
           (context.portal,lugar.contenido,html_quote(lugar.title))+"""<br>""")
for carpeta in lugar.objectValues('Folder'):
   if request.AUTHENTICATED_USER.has_permission('View',carpeta):
           lista.insert(9999,   """    <img   src="/%s/Skins/%s/i%s.gif"
width="10" height="10">""" % \
           (context.portal,context.Skins.buena,carpeta.icono)
            +
```

```
           """ <a href="/%s/index_html?p1=%s;p2=%s" target="_top">%s</a>""" % \
           (context.portal,html_quote(carpeta.indice),html_quote(carpeta.contenido),
html_quote(carpeta.title))+"""<br>""")
   nueva=[]
   if carpeta.getId()!='Mas' and carpeta.getId()!='Tablon':
     nueva=context.buscaRec(carpeta,'        ')
   if nueva!=[]:
    for item in nueva:
     lista.insert(9999,item)

return lista
```

/demo/Mapa/buscaRec

```
#parametros: lugar (carpeta actual) y tab (nivel de tabulación)
from Products.PythonScripts.standard import html_quote
request = context.REQUEST
lista=[]
for carpeta in lugar.objectValues('Folder'):
 if request.AUTHENTICATED_USER.has_permission('View',carpeta):
   lista.insert(9999, """%s<img src="/%s/Skins/%s/i%s.gif" width="10" height="10">""" % \
           (tab,context.portal,context.Skins.buena,carpeta.icono)
           +
           """ <a href="/%s/index_html?p1=%s;p2=%s" target="_top">%s</a>""" % \
           (context.portal,html_quote(carpeta.indice),html_quote(carpeta.contenido),
html_quote(carpeta.title))+"""<br>""")
   nueva=context.buscaRec(carpeta,tab+'    ')
   if nueva!=[]:
       for item in nueva:
        lista.insert(9999,item)
return lista
```

8. *Script* Python (`/demo/getTitulares`) que retorna una lista que contiene los objetos asociados a las noticias publicadas en el lugar y/o la sección que se indica.

```
request = context.REQUEST
lista=[]
for titular in container.Perfil.Noticias.Publicar.News.objectValues('Folder'):
 if titular.visible=='S':
 # //como no se eliminar
  if (donde=='portada'):
   if (titular.prioridad=='A'):
       if (titular.privada=='N') or
               (request.AUTHENTICATED_USER.has_role(['profesor','Manager'])) :
         lista.insert(0,titular)
  elif (donde=='tablon' and seccion==''):
     if (titular.seccion!='Alumnado' or alumno!=''):
       if (titular.privada=='N') or
                 (request.AUTHENTICATED_USER.has_role(['profesor','Manager'])) :
         lista.insert(0,titular)
  elif (donde=='tablon' and seccion!=''):
     if (titular.seccion==seccion):
       if (titular.privada=='N') or
                 (request.AUTHENTICATED_USER.has_role(['profesor','Manager'])) :
         lista.insert(0,titular)
  elif (donde=='personal'):
       if (request.AUTHENTICATED_USER.has_role(['Manager'])):
         lista.insert(0,titular)
       elif (titular.autor==request.AUTHENTICATED_USER.getId()):
             lista.insert(0,titular)
return lista
```

9. *Script* Python (`/demo/sendWebMail`) que envía un mensaje instantáneo a. Se incluye el código del método SQL (`/demo/insertMail`) que realiza la inserción en la base de datos.

```
#parámetros: texto (el texto del mensaje ) y destino (usuario destino)
from DateTime import DateTime

fecha='%s/%s/%s' % (DateTime().dd(),DateTime().mm(),DateTime().yy())
origen=context.getUsuario()
context.insertMail(texto=texto,destino=destino,fecha=fecha,origen=origen)

return
```

`/demo/insertMail`

```
# Parámetros: texto, destino, fecha y origen
insert into mensajes values (
'<dtml-var texto>','<dtml-var destino>','N','<dtml-var fecha>','<dtml-var origen>')
```

10. Plantilla Zope (`/demo/ultimaHora`) que las novedades de la página de inicio.

```
<html>

<head>
<title>Portada</title>
<base target="_self">
<link rel="STYLESHEET" type="text/css" href="content.css">

</head>
<body ><p> </p>

<div align="center">
  <center>
  <table class="normal" border="0" cellpadding="0" cellspacing="0">
    <tr class="titulotabla">
      <td height="30">
      <p align="center"><b>NOVEDADES</td>
    </tr>
    <tr>
     <td><br><td>
    </tr>

    <span tal:repeat="noticia python:here.getTitulares('portada')">
    <tr tal:define="global vez repeat/noticia/number"
            tal:condition="python:vez<=here.maxPortada">
      <td>
              <b tal:content="python:'[%s]'% here.getFechaddmmaa(noticia)">11-05-2004
       </b>
        <a href="kkkk" target="_top" tal:content="noticia/title"
              tal:define="url noticia/absolute_url"
              tal:attributes="href python:here.getDestino('Indice',url)">titular</a>
       </td>
    </tr>
      <tr  ><td  align="center"   tal:condition="python:vez<here.maxPortada"><br>-----------
<br><br></td></tr>

    </span>
  </table>
  </center>
</div>


</body>
</html>
```

11. *Script* Python (`/demo/changePassword`) que cambia la contraseña del usuario registrado.

```
#parámetros: user (usuario) y pwd1 (nueva contraseña)
request = context.REQUEST
RESPONSE =  request.RESPONSE
lista=[]
if request.AUTHENTICATED_USER.has_role(['Manager']):
 lista.insert(0,'Manager')
if request.AUTHENTICATED_USER.has_role(['alumno']):
 lista.insert(0,'alumno')
if request.AUTHENTICATED_USER.has_role(['profesor']):
 lista.insert(0,'profesor')
if request.AUTHENTICATED_USER.has_role(['equipo']):
 lista.insert(0,'equipo')
if request.AUTHENTICATED_USER.has_role(['jefedpto']):
 lista.insert(0,'jefedpto')

context.REQUEST.set("domains", [])
context.REQUEST.set("name", user)
context.REQUEST.set("roles", lista)
context.REQUEST.set("password", pwd1)
context.REQUEST.set("confirm", pwd1)
lugar = context.restrictedTraverse('/%s' % context.portal)
context.acl_users.manage_users(submit="Change", REQUEST=context.REQUEST)
return
```

12. *Script* Python (`/demo/Config/Minimo/Logos/upLoadImagen`) que publica los logotipos en la configuración del sitio Web.

```
#parametros: fichero (el archivo) y tipo (el tipo de logo)
if fichero.filename=='' or tipo=='':
 return context.REQUEST.RESPONSE.redirect("respUpLoad?error=si")

# crear el fichero
lugar = context.restrictedTraverse('/%s/imagen' % context.portal)
if tipo=='ca':
 for img in lugar.objectValues():
   if img.getId()=='logocam':
     lugar.manage_delObjects(['logocam'])
 id='logocam'
elif tipo=='centro':
 for img in lugar.objectValues():
   if img.getId()=='logocentro':
     lugar.manage_delObjects(['logocentro'])
 id='logocentro'
else:
 for img in lugar.objectValues():
   if img.getId()=='imgportada':
     lugar.manage_delObjects(['imgportada'])
 id='imgportada'

lugar.manage_addProduct['OFSP'].manage_addImage(id, file=fichero)
return context.REQUEST.RESPONSE.redirect("respUpLoad")
```

13. *Script* Python (`/demo/Perfil/Noticias/Publicar/insNoticia`) que realiza la publicación de una noticia. Se incluye el *Script* Python que permite adjuntar un archivo a la noticia (`/demo/Perfil/Noticias/upLoadFile`).

```
#parametros: contenido,titular,mas,dirweb,privada,seccion,prioridad, REQUEST=None
""" Creación de una notica
-1. Comprobar que titular no es vacío
0.- Mirar cual será el identificador de la carpeta
1.- Crear la carpeta
2.- Añadir propiedades
3.- Crear el archivo con el contenido
"""
import random

#-1.
if titular=='':
 return context.REQUEST.RESPONSE.redirect("respPublicar?error=1")

# 0.
nuevovalor=0
# lugar=container.News
lugar = context.restrictedTraverse('/%s/News'%context.portal)
nuevovalor=lugar.elementos+1
lugar.manage_changeProperties(elementos=nuevovalor)
aleat=random.randint(1000000,9999999)
id='not_%d_%d' % (nuevovalor,aleat)
#1.
lugar.manage_addProduct['OFSP'].manage_addFolder(id, title=titular)
#2.

if dirweb=='http://':
 dirweb=''
nueva=getattr(lugar, id)
nueva.manage_addProperty('masinfo', mas, 'string')
nueva.manage_addProperty('dirweb', dirweb, 'string')
nueva.manage_addProperty('privada', privada, 'string')
nueva.manage_addProperty('prioridad', prioridad, 'string')
nueva.manage_addProperty('seccion', seccion, 'string')
nueva.manage_addProperty('autor', context.getUsuario(), 'string')
nueva.manage_addProperty('contenido', contenido, 'string')

return context.REQUEST.RESPONSE.redirect("respPublicar?noticia=%s" % (id))
```

`/demo/Perfil/Noticias/upLoadFile`

```
#parametros: fichero (el contenido del archivo), noticia(el identificador de la noticia
# al que está adjunto) y titulo (el titular que se visualizará con el archivo)

from Products.PythonScripts.standard import url_quote
if fichero.filename=='' or titulo=='' or
       context.tamanoMaximo(context.REQUEST.CONTENT_LENGTH)==1:
 return context.REQUEST.RESPONSE.redirect("respUpLoad?error=%s" % (noticia) )

# crear el fichero
lugar=container.News
lugar=getattr(lugar, noticia)
id='adjunto_%d' % len(lugar.objectIds())

# if fichero.content_type=='image/gif' or fichero.content_type=='image/jpeg' or
       fichero.content_type=='image/png':
# lugar.manage_addProduct['OFSP'].manage_addImage(id, title=titulo, file=fichero)
# else:

lugar.manage_addProduct['OFSP'].manage_addFile(id, title=titulo, file=fichero)
return context.REQUEST.RESPONSE.redirect("respUpLoad?noticia=%s" % (noticia) )
```

14. Plantilla Zope (`/demo/Indice/Tablon/Todas/index_html`) que muestra el tablón de anuncios con los titulares de las últimas noticias.

```
<html>
  <head>
    <title tal:content="template/title">The title</title>
<link rel="STYLESHEET" type="text/css" href="content.css">

  </head>
  <body>

    <span class="ruta"  tal:content="structure container/darruta">ruta de
naavegacion</span>
<div align="right">
  <table class="addfavor" border="0"
     tal:condition="python:here.getRol('profesor')=='true'">
    <tr>
      <td>»» <a href="www" tal:define="aqui here/absolute_url; donde
      python:here.substring(aqui)"
     tal:attributes="href python:'/%s/addFavor?nombre=%s&p1=%s&p2=%s' %
        (here.portal,here.title,here.indice,donde)">Agregar a mis favoritos </a>««
    </td>
    </tr>
  </table>
</div>
<br>

<table class="pequena" width="100%">
 <tr>
   <td align="center" width="25%">
        <form action="../Secciones/verSeccion" method="POST">
          input type="submit" name="B1" value="Elegir Seccion" style="font-family: Arial;
             font-size: 8 pt"><br>
     <select name="seccion" size="1" style="font-family: Arial; font-size: 9 pt; border-
          style: solid">
                <option selected value="Comunidad Educativa">Comunidad Educativa
                <option value="Dirección">Dirección
                <option value="Jefatura de Estudios">Jefatura de Estudios</option>
                <option value="Secretaría">Secretaría</option>
                <option value="ESO">ESO</option>
                <option value="Bachillerato">Bachillerato</option>
                <option value="Formación Profesional">Formación Profesional</option>
                <option value="Garantía Social">Garantía Social</option>
                <option value="Extraescolares">Extraescolares</option>
                <option value="AMPA">AMPA</option>
                <option value="Alumnado">Alumnado</option>
               </select>
         </form>

   </td>
   <td width="50%" align="center">
<h1> Noticias Publicadas
</h1>
   </td>
   <td align="center">
        <form action="../Buscar" method="POST">
           <input type="text" name="cadena" style="font-family: Arial; font-size: 9 pt;
             border-style: solid"><br>
           <input type="submit" name="B1" value="Buscar" style="font-family: Arial; font-
              size: 8 pt">
</form>


   </td>
 </tr>
</table>


<table  class="titulotabla" border="0" cellpadding="5" cellspacing="0" width="100%">
    <tr>
```

```
      <td align="center" width="15%">Fecha</td>
      <td align="center" width="70%">Titular de la Publicación</td>
      <td align="center" width="15%">
        Archivos</td>
    </tr>
</table>


<table class="normal" border="0" cellspacing="0" cellpadding="10" width="100%"
    tal:define="noticias python:here.getTitulares('tablon');
         start request/start|python:0;
         batch python:modules['ZTUtils'].Batch(noticias,size=here.maxNoticias,start=start);
         global previous python:batch.previous;global next python:batch.next">

   <span tal:repeat="noticia batch">
<tr class="ll"
  tal:attributes="class python:test(path('repeat/noticia/even'), 'gris', 'blanco')">
   <td align="center" width="15%"><img border="0" src="../imagen/flecha.gif" width="12"
          height="12"> 
          <a href="noticia1"
               tal:attributes="href noticia/absolute_url"
               tal:content="python:here.getFechaddmmaa(noticia)">la fecha</a>
      </td>
      <td width="70%" tal:content="noticia/title">titulo     </td>
      <td align="center">
         <i tal:content="python:here.hayAdjuntos(noticia)">Si</i>
      </td>
     </tr>
     </span>

  </table>
<br>
<p align="center">
<a tal:condition="previous" tal:attributes="href
    string:${request/URL0}?start:int=${previous/first}" href="previous_url">Anterior</a>
<a tal:condition="next" tal:attributes="href
    string:${request/URL0}?start:int=${next/first}" href="next_url">Siguiente</a>
</p>

<br>
<li class="mensaje"> Las noticias que pertenecen a la sección de Alumnado se visualizan en
la opción <i>Elegir Sección</i></li>

</body>
</html>
```

15. Plantilla Zope (`/demo/Perfil/Jefe/Registrar/addProf`) que añade un profesor al portal. Se incluye el *Script* python que añade el usuario al sistema Zope.

```
<html>
  <head>
    <title tal:content="template/title">The title</title>
<link rel="STYLESHEET" type="text/css" href="content.css">

  </head>
  <body>
<span tal:condition="python:request.user=='' or request.pwd1=='' or request.prof=='' or
request.pwd1!=request.pwd2">
  <script>window.open("respAddProf?error=1","contenido");</script>
</span>

<span tal:condition="python:not (request.user=='' or request.pwd1=='' or request.prof==''
or request.pwd1!=request.pwd2)">

 <table tal:repeat="item python:here.existeUser(usuario=request.user)">
  <tr tal:define="global numero string:${item/numero}">
   <td
tal:condition="python:numero>'0'"><script>window.open("respAddProf?error=1","contenido");</
script></td>
```

```
   <td tal:condition="python:numero=='0'">
     <span   tal:define="nada   python:here.insProf(user=request.user,   pwd1=request.pwd1, prof=request.prof,dpto=request.dpto)"></span>
     <i tal:define="dd python:here.sendWebMail(texto='Acabas de pasar a formar parte del portal, esperamos que obtengas todo lo que esperas ...',destino=request.user)"></i>

<br><br>
<p class="ok">
 El usuario se ha registrado correctamente.
</p>

  </td>
  </tr>
 </table>
</span>


 </body>
</html>
```

/demo/Perfil/insProf

```
lugar = context.restrictedTraverse('/%s/Personal' % context.portal)
lugar.manage_addProduct['OFSP'].manage_addFolder(user, title='Página de %s' %user)

nueva=getattr(lugar, user)
nueva.manage_addProperty('elementos', 0, 'long')
nueva.manage_addProperty('f1', '', 'string')
nueva.manage_addProperty('f2', '', 'string')
nueva.manage_addProperty('f3', '', 'string')
nueva.manage_addProperty('f4', '', 'string')
nueva.manage_addProperty('f5', '', 'string')
nueva.manage_addProperty('p11', '', 'string')
nueva.manage_addProperty('p12', '', 'string')
nueva.manage_addProperty('p21', '', 'string')
nueva.manage_addProperty('p22', '', 'string')
nueva.manage_addProperty('p31', '', 'string')
nueva.manage_addProperty('p32', '', 'string')
nueva.manage_addProperty('p41', '', 'string')
nueva.manage_addProperty('p42', '', 'string')
nueva.manage_addProperty('p51', '', 'string')
nueva.manage_addProperty('p52', '', 'string')
nueva.manage_addProperty('contenido', '/%s/Personal/%s' %(context.portal,user), 'string')

context.updateProf(apellidos=prof,user=user,departamento=dpto)

context.REQUEST.set("domains", [])
context.REQUEST.set("name", user)
context.REQUEST.set("roles", ["profesor"])
context.REQUEST.set("password", pwd1)
context.REQUEST.set("confirm", pwd1)
lugar = context.restrictedTraverse('/%s' % context.portal)
context.acl_users.manage_users(submit="Add", REQUEST=context.REQUEST)
```

ANEXO II.- INFORMACIÓN QUE OFRECE *GASPWIES*.

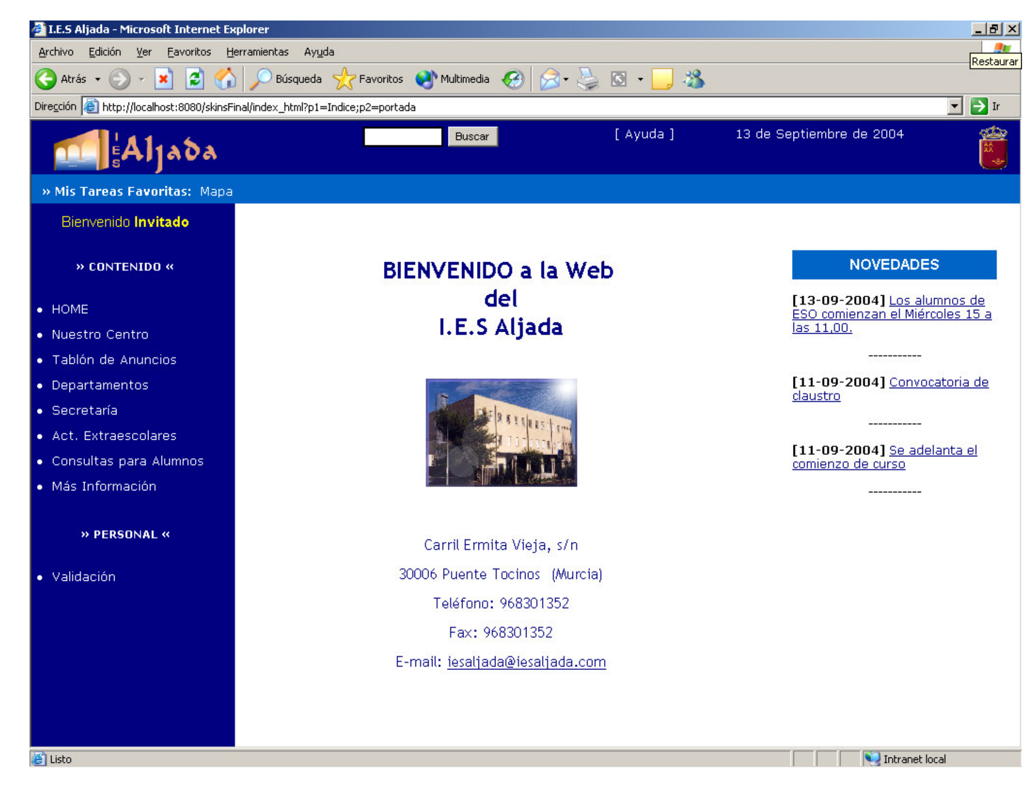


Figura II.1. Página de inicio con navegación en modo texto.



Figura II.2. Página de inicio con navegación en modo gráfico.

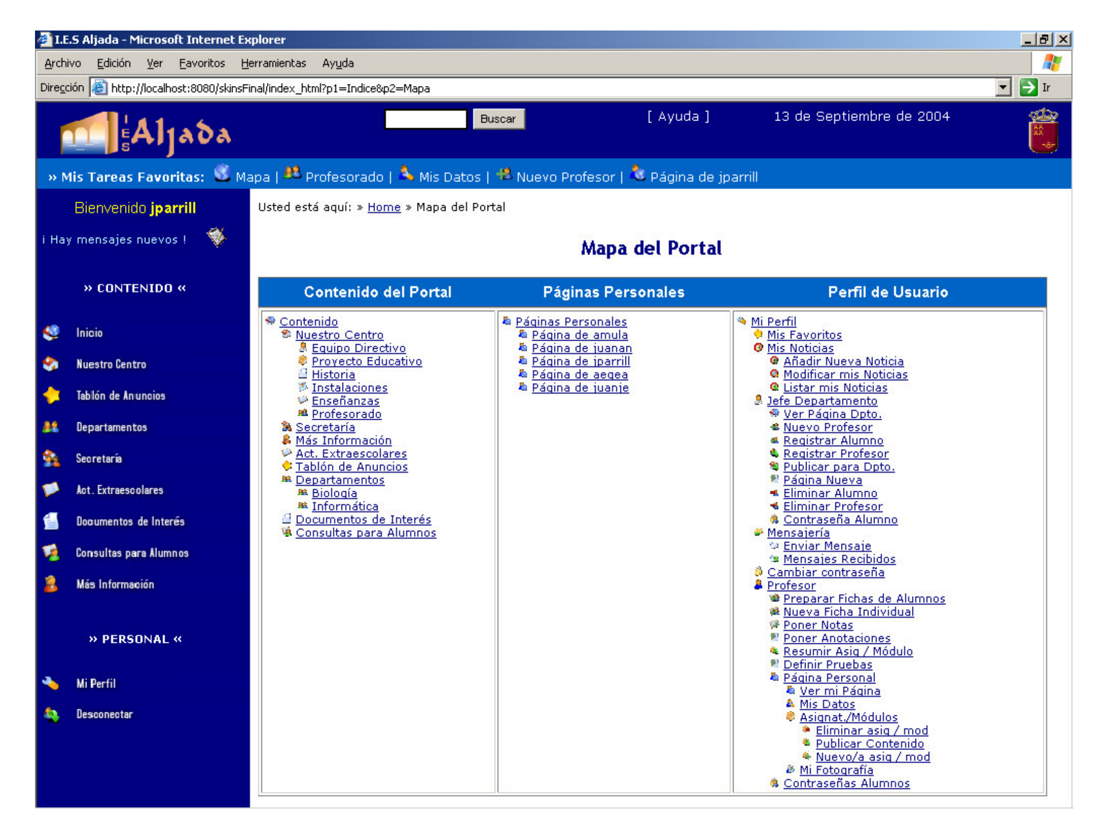


Figura II.3. Mapa del sitio Web.

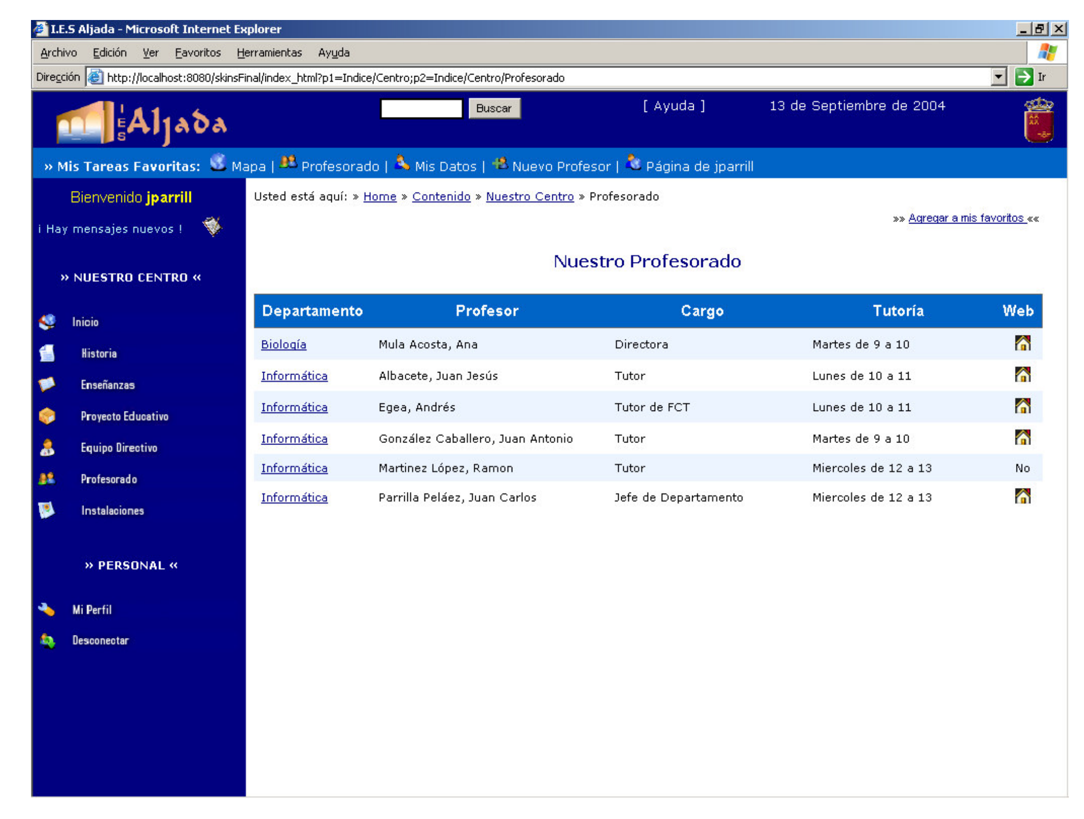


Figura II.4. Relación del profesorado que forma el claustro de profesores del Centro.

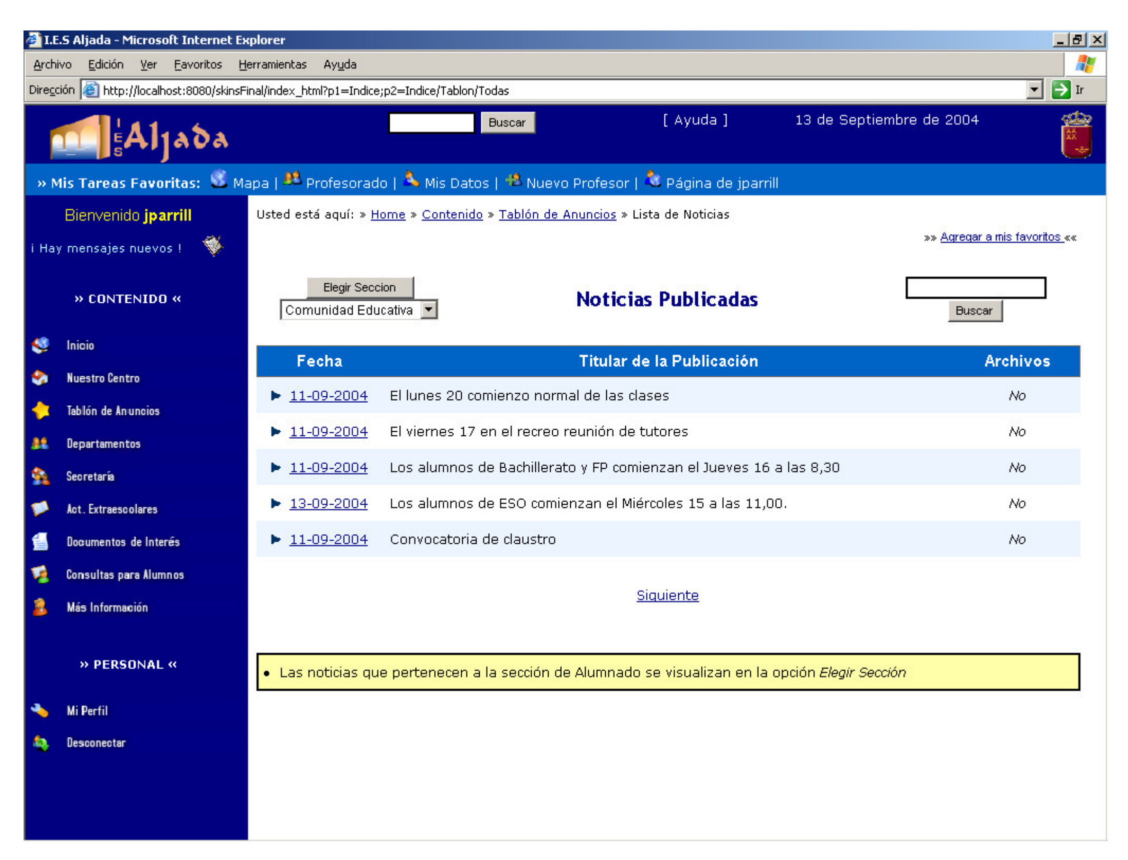


Figura II.5. Tablón de anuncios con las últimas noticias

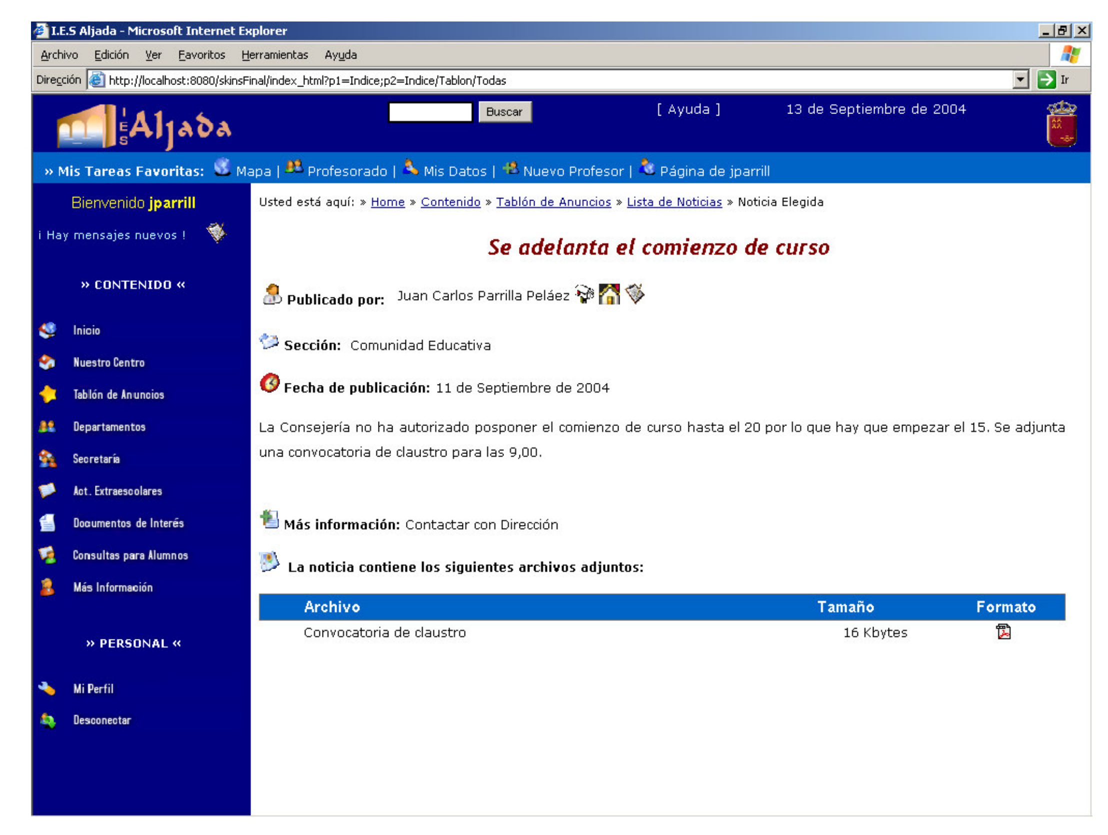


Figura II.6. Contenido de una noticia completa.



Figura II.7. Página de un Departamento Didáctico.



Figura II.8. Página personal de un profesor.

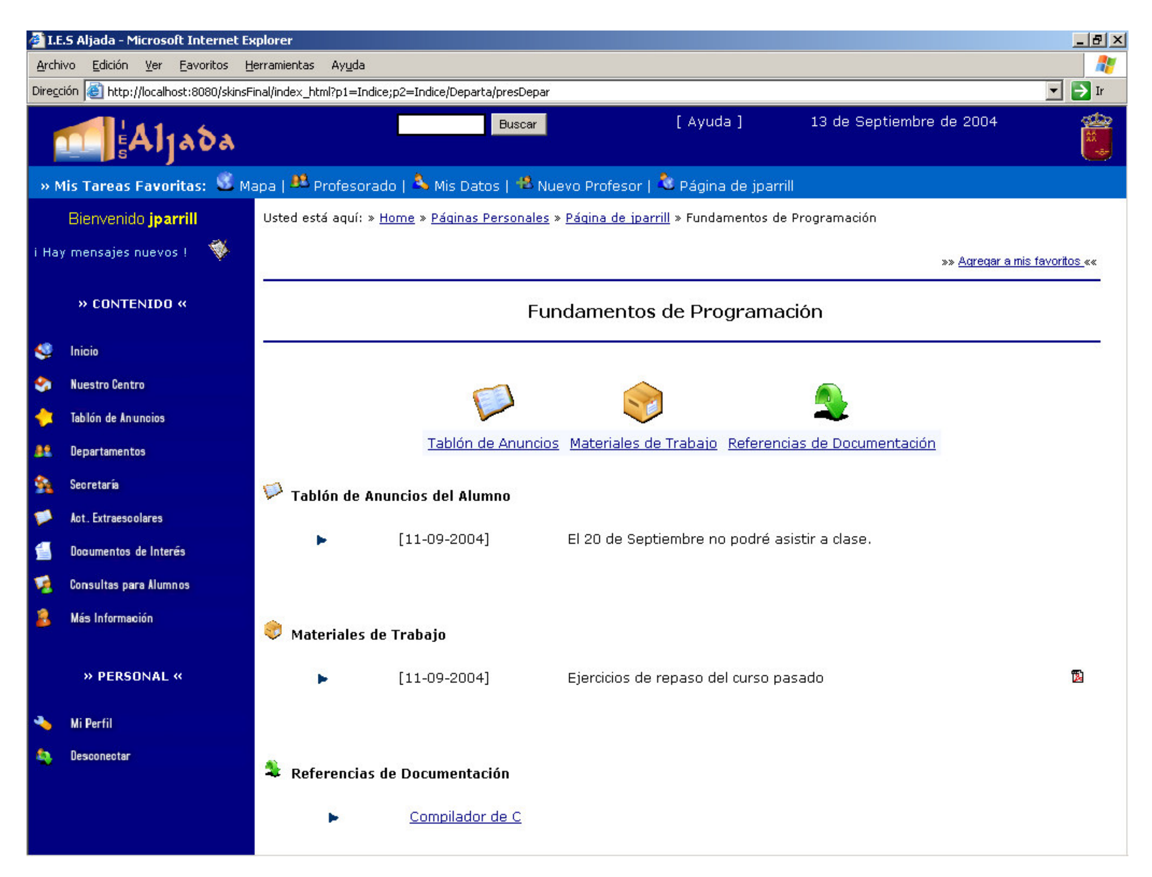


Figura II.9. Página de una materia.

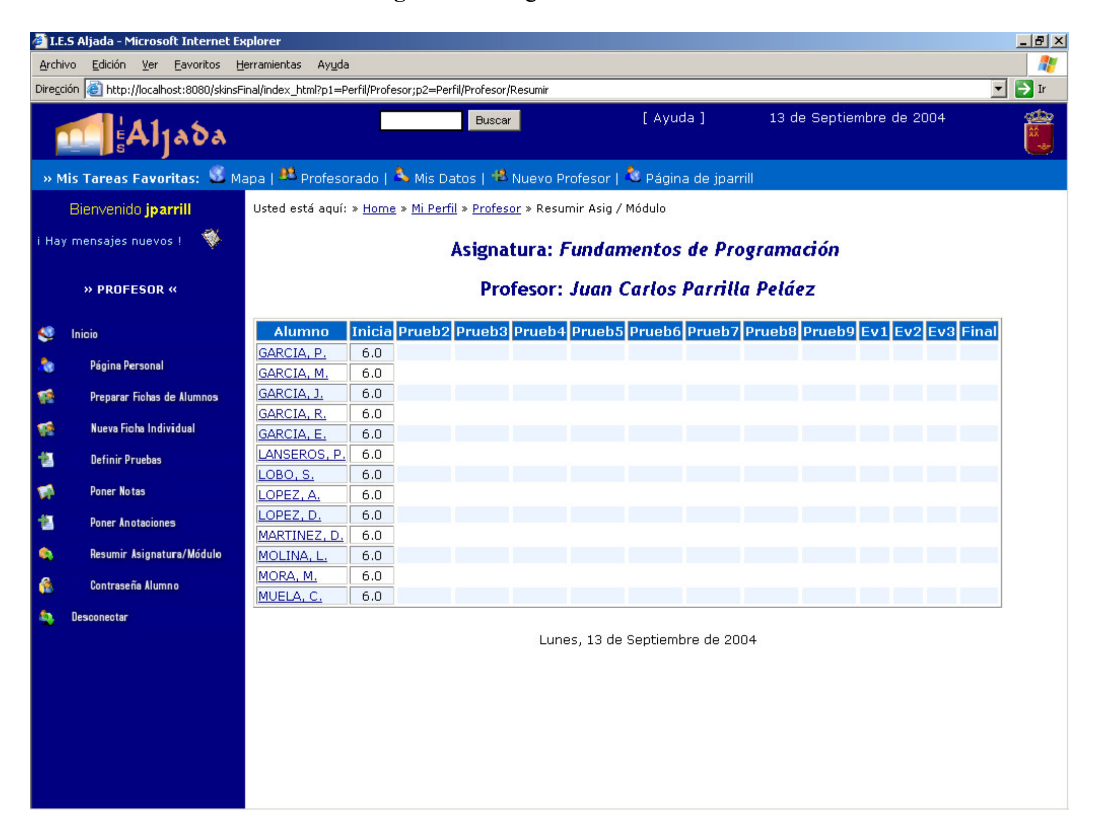


Figura II.10. Visión general del seguimiento de una materia.

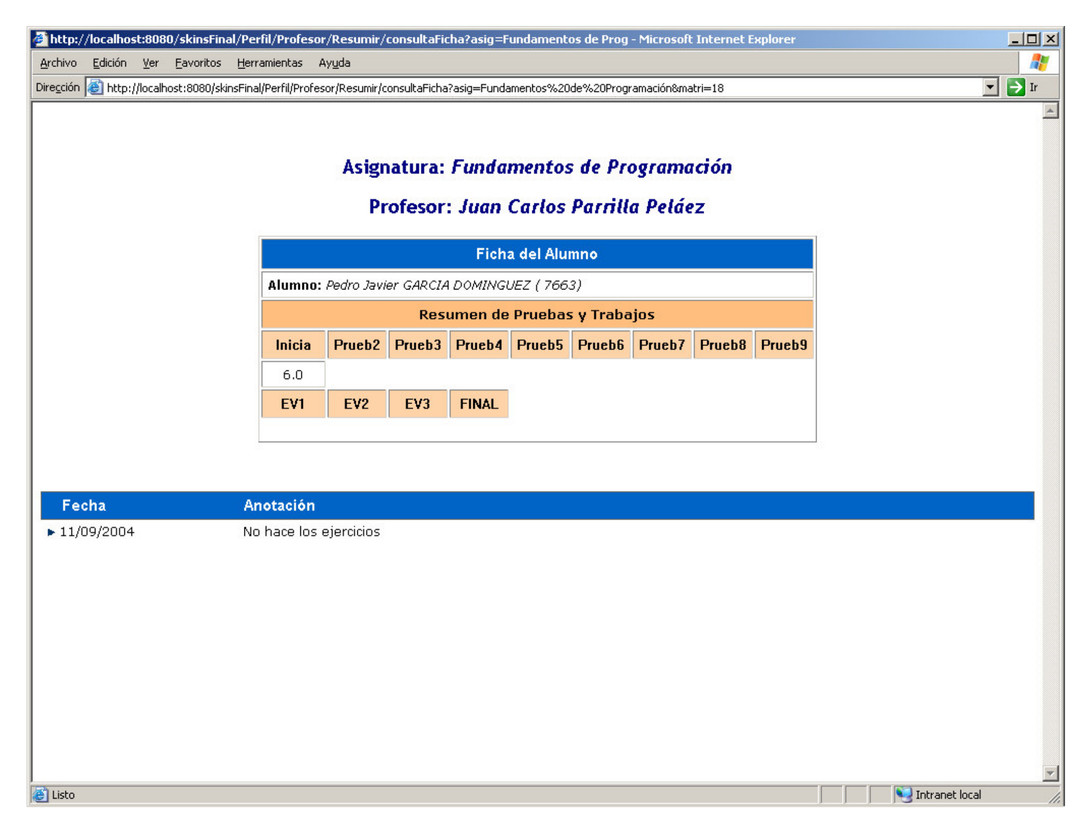


Figura II.11. Ficha de un alumno.

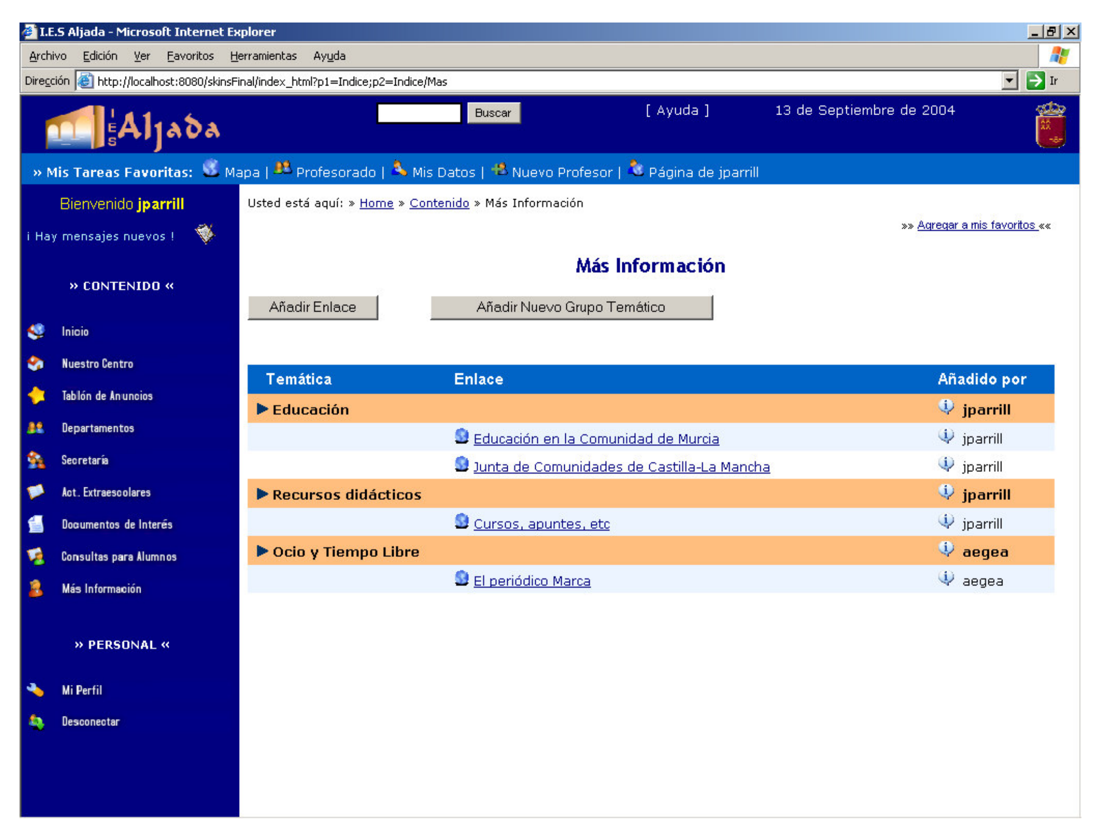


Figura II.12. Enlaces más interesantes a juicio de la Comunidad Educativa.

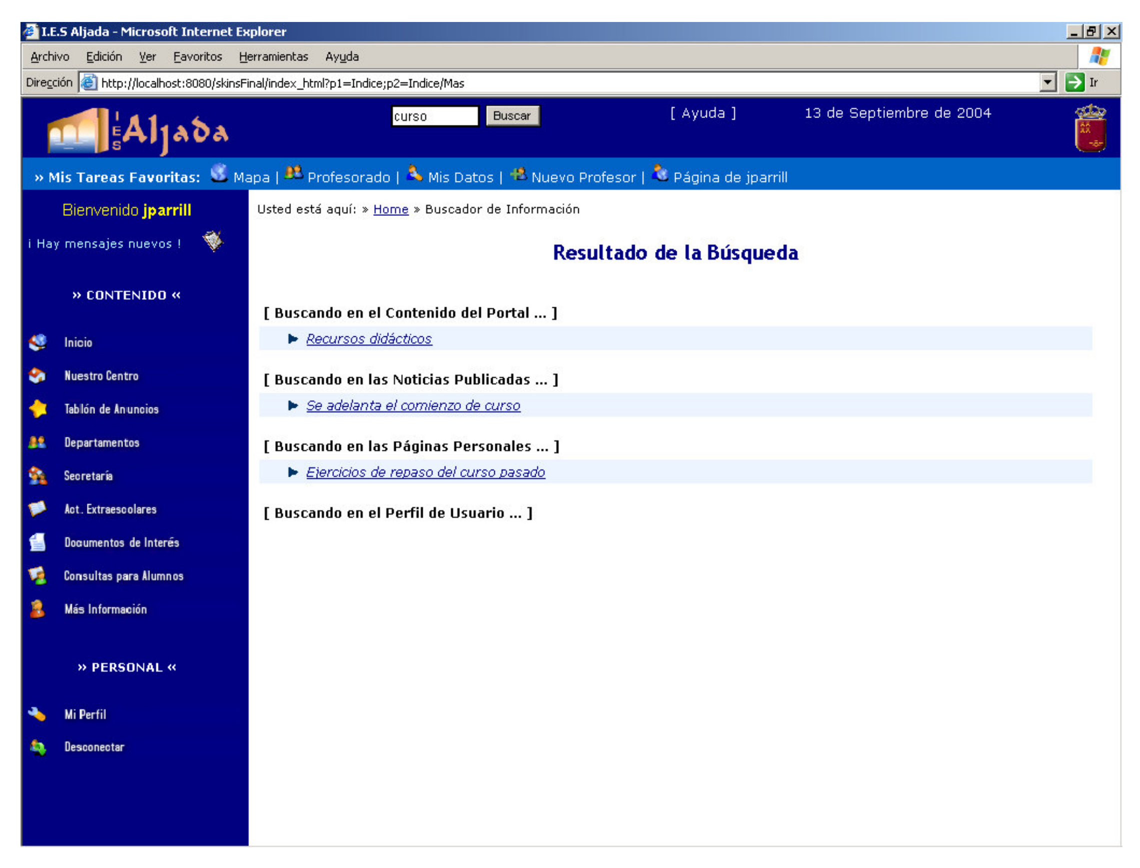


Figura II.13. Resultado de una búsqueda de información.